Dudley Knox Library, NPS
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION
OF SOFTWARE PROTOCOL IN VAX/VMS
USING ETHERNET LOCAL AREA NETWORK

by

Thawip P. Netniyom
June 1983

Thesis Advisor: U.R. Kodres

Approved for public release; distribution unlimited

T210147

Dudley Knox Library, NPS
Monterey, CA 93943

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: Design and Implementation of Software Protocol in VAX/VMS Using Ethernet Local Area Network	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, June 1983 / 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)*: Thawip P. Netniyom	8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: June 1983 / 13. NUMBER OF PAGES: 118
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: Unclassified / 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Ethernet, Unibus, Multibus, Network, Protocol Hierarchies
DECnet, ISO Reference Model

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This thesis presents the design and implementation of a software protocol for the VAX-11/780, under the VMS operating system, to allow message and file transfer to and from the INTELLEC MDS system, under CP/M-80 operating system, via the Ethernet local area network.

The design of this software protocol is based on the protocol hierarchies where the network is organized as a series of layers

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

or levels, each one build upon its predecessor. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

With this design concept, the desired software protocol will be transportable in the sense that it can be used by different computer systems with minimal modifications.

The Ethernet local area network is also designed in this same highly structured way.

Approved for public release; distribution unlimited.

Design and Implementation of Software Protocol in VAX/VMS Using Ethernet Local Area Network

by

Thawip P. Netniyom
Second Lieutenant, Royal Thai Army
B.S., Philippine Military Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

T16z.
1424
c.1

ABSTRACT

This thesis presents the design and implementation of a software protocol for the VAX-11/780, under the VMS operating system, to allow message and file transfer to and from the INTELIEC MDS system, under CP/M-80 operating system, via the Ethernet local area network.

The design of this software protocol is based on the protocol hierarchies where the network is organized as a series of layers or levels, each one build upon its predecessor. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

With this design concept, the desired software protocol will be transportable in the sense that it can be used by different kinds of computer systems with minimal modifications.

The Ethernet local area network is also designed in this same highly structured way.

TABLE OF CONTENTS

I. INTRODUCTION 9
A. DISCLAIMER 9
B. GENERAL DISCUSSION 9
C. BACKGROUND 10
D. STRUCTURE OF THE THESIS 11

II. BASIC DESIGN CONCEPTS 13
A. PROTOCOL FOR LOCAL AREA NETWORK 13
1. Background 13
2. Application subsystem perspective of elementary protocols 16
B. PROTOCOL HIERARCHIES 20
C. DESIGN ISSUES FOR THE LAYERS 22
D. THE ISO REFERENCE MODEL 24

III. ETHERNET LOCAL AREA NETWORK 30
A. OVERVIEW 30
B. CONCISE ETHERNET SPECIFICATION 31
1. Packet Format 31
2. Control Procedure 33
3. Channel Encoding 34
4. Data Rate 35
5. Carrier 35
6. Coax Cable 35
7. Coax Connectors and Terminators 36
8. Transceiver 37
C. NI1010 UNIBUS ETHERNET COMMUNICATIONS CONTROLLER 37
1. Features 37

2. Description 38
3. Ethernet data link layer functions 38
4. Ethernet physical layer functions 43
5. Performance 44
6. Extensive Diagnostic Features 44
7. Network statistics 45

IV. DETAILED DESIGN AND IMPLEMENTATION 47
A. DESIGN ISSUES CONSIDERATION 47
B. NS2030 DEVICE DRIVER 48
1. Overview 48
2. Program interfaces to NIDRIVER 50
C. DESIGNING PROCEDURE 58
1. Steps in developing the application layer 59
2. Method to Overcome Frame Sequencing . . . 60
D. IMPLEMENTATION 62

V. CONCLUSION 63

APPENDIX A: VAX/VMS SYSTEM SERVICE ROUTINES 67

APPENDIX B: SOURCE CODE FOR EXPERIMENTS 81

APPENDIX C: ETHERNET SOFTWARE PROTOCOL SOURCE CODE . . . 93

APPENDIX D: VAX/VMS-MDS ETHERNET LOCAL COMMUNICATION NETWORK USER MANUAL 106

APPENDIX E: SYMBOLIC NAME FOR NI1010 CONTROLLER COMMAND CODE 112

APPENDIX F: NIDRIVER FUNCTION AND STATUS CODE SUMMARY 114

LIST OF REFERENCES 116

INITIAL DISTRIBUTION LIST 117

LIST OF TABLES

I. Type Field Protocol: (All in Hexadecimal) 61

LIST OF FIGURES

2.1 Local Area Network Topologies 15
2.2 Layers, Protocols, and Interfaces 21
2.3 ISO Reference Model 24
3.1 Contention, Transmission, and Idle States . . . 30
3.2 Ethernet Packet Format 31
3.3 Data Rate Scheme 34
3.4 Coax Cable, Connectors, and Transceivers 36
3.5 Ethernet Architecture and Implementation 39
3.6 Ethernet Frame Format 40
3.7 NI1010 Functional Diagram 45
4.1 NI1010 and ISO Reference Model 49
4.2 Transmit Packet Format 55
4.3 Receive Packet Format 56
4.4 Mapping between ISO Model and DECNET 58

I. INTRODUCTION

A. DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark:

INTEL Corporation, Santa Clara, California
- INTELLEC MDS
- Multibus

DIGITAL Research, Pacific Grove, California
- CP/M-80

INTERLAN Corporation, Chelmsford, Massachusetts
- NI1010 Unibus Ethernet communications controller board
- NI3010 Multibus Ethernet communications controller board
- NS2030 VMS device driver and NI1010 diagnostic program

DIGITAL Equipment Corporation, Maynard, Massachusetts
- VAX-11/780 Mini computer
- VAX/VMS operating system

B. GENERAL DISCUSSION

This thesis presents the design and implementation of software protocol for the VAX-11/780, under VMS operating system, to allow message and file transfer to and from INTELLEC MDS system, under CP/M-80 operating system, via Ethernet local area network.

The Ethernet board is the product of Interlan company which has produced the hardware and software technology needed to connect several makes of computers to the network. All communications between host computers are made through a coaxial cable which has a 10 megabit per second data communications rate. The Ethernet design is based on a highly structured protocol where the network is organized as a series of layers or levels, each one build upon its predecessor. The purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

This concept of design yields a lot of advantages in developing software protocol such that it can be used by different computer systems with the minimum of modifications. It is anticipated that the software protocol designed will be general in nature in order to be used with other computers using different operating systems with minor changes. The specific goals of this thesis are discussed in the next section concerning the background of the project.

C. BACKGROUND

The AEGIS weapons system simulation project currently being conducted at the Naval Postgraduate School is attempting to determine the feasibility of replacing the larger and relatively expensive main frame computer, the AN/UYK-7, with a system of 16 or 32 bit micro computers. Several significant real-time functions of the AEGIS weapons system are to be duplicated with associated data, inputs, timing, and supporting function so that a test example can be examined whose performance emulates that of the actual system. [Ref. 1.]

Since the AEGIS weapons system simulation project involves many micro computers and since all of them must be real-time system, the speed of data communications between any two computers is very crucial. Thus, a high-speed means of communication is needed.

Ethernet local area network is considered to be the solution because of its capability of permitting 10 megabit per second data communications between stations seperated by up to 2500 meters. Two NI3010 Multibus Ethernet communication controller boards and a NI1010 Unibus Ethernet communication controller boards were purchased and implemented in INTELIEC MDS systems and VAX-11/780, under the VMS operating system, respectively. An NS2030 device driver and an NI1010 diagnostic program are also implemented in the VAX.

The specific goals of this thesis are:

1.To design and implement software protocol on the VAX/VMS such that it would be able to communicate (transfer messages and files) with the INTELLEC MDS system, under the CP/M operating system.

2.To be used as a guideline in designing software protocols, especially when there is a need to communicate between VAX/VMS and other systems which use different operating systems such as ISIS-II.

D. STRUCTURE OF THE THESIS

Chapter I presents a general discussion of the larger, ongoing effort of which this thesis is a part. It also gives a general discussion of the background work of the AEGIS weapons system simulation project and the need of the Ethernet local area network which leads to this thesis research.

Chapter II addresses the overall design philosophy of the software protocol which will be used in the VAX/VMS. Protocol for a local area network is discussed in detail to include the relationship to the ISO open system interconnection model. The design issues for the layers are enumerated to be a step-by-step guide in designing the protocol.

Chapter III discusses the Ethernet Local Area Network, concise Ethrnet specification, and the NI1010 Unibus Ethernet Controller Board.

Chapter IV explains the detailed design of the software protocol to include the use of the NS2030 Device Driver, how Ethernet Board and the Device Driver take care of the design issues for the layer mentioned in Chapter II, and steps in developing the application layer protocol.

Chapter V summarizes the testing of the implemented software protocol and describe its capabilities. Suggestions are also given for future research and modification.

II. BASIC DESIGN CONCEPTS

A. PROTOCOL FOR LOCAL AREA NETWORK

1. Background

A set of protocols specifies how nodes can communicate over networks. Protocols are the procedures and conventions used to regiment the event progression required for orderly, mutually understood interaction between processes. Protocols are developed to satisfy qualitative and quantitative requirements for process interconnection. A primary qualitative requirement is the "useful" work (i.e., functionally) to be provided by the protocol. Other qualitative requirements include:

a). flexibility (to accommodate new uses and features)

b). completeness (to properly respond to all relevant network conditions)

c). deadlock avoidance/backout mechanisms

d). synchronization mechanisms (for interprocess control)

e). error detection and recovery

f). buffer overflow avoidance

g). message sequencing assurance

h). duplicate message detection and recovery

i). permeance (to implement the protocol uniformly through the local area network)

j). priority mechanisms

k). accounting mechanisms

l).security mechanisms

m).message delivery guarantees

n).data code/format transformations

o).computer equipment feature compatibility

p).operating system feature compatibility

q).communications network feature compatibility

Not all of these requirements are of equal importance for each protocol implementation. For example, a protocol might inhabit a local area network node environment configured into any of the following topologies or hybrids of these topologies:

a).star topology - a centralized topology in which lines converge to a central point or points (see Figure 2.1);this topology is also called hierarchical or tree.

b).mesh topology - nodes are connected in an arbitrary pattern; each node can have multiple paths to other nodes.

c).ring topology - the communications path is a loop with each node connected to exactly two other nodes in a given loop.

d).bus topology - the nodes are connected along line segments; this topology is commonly found in Local area networks with a shared transmission channel such as a cable-bus.

Typically, routing protocols for mesh topologies are much more complex than routing protocols for the other topologies. Thus, other protocol requirements, such as message delivery guarantees and message sequencing assurance, may be influenced by the topology chosen.

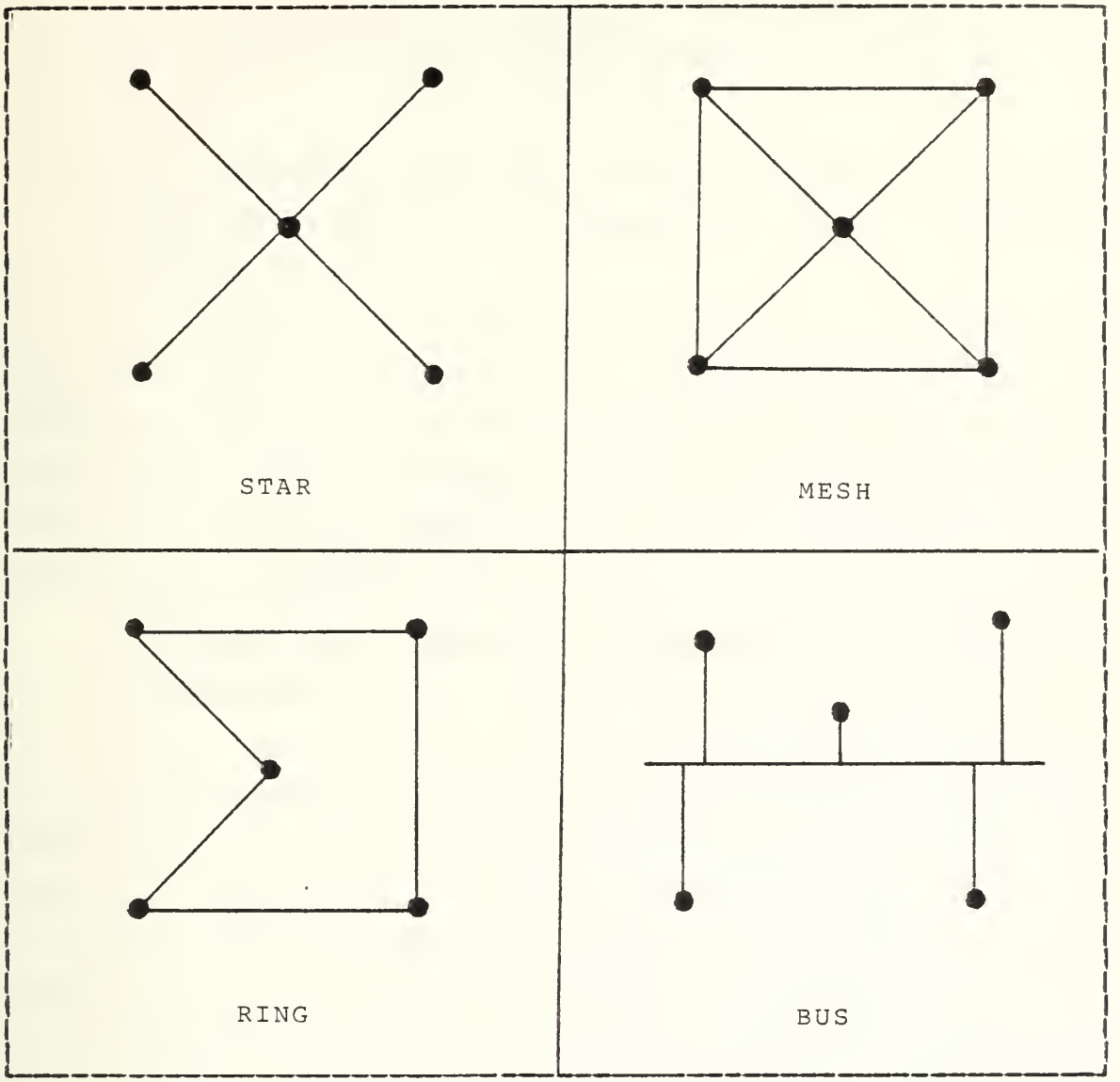


Figure 2.1 Local Area Network Topologies.

Quantitative requirements for protocols include:

a) throughput - the volume of information that must be transferred during the peak period. This volume is usually characterized by mean message length in octets, the distribution of message lengths, and by the arrival rate of messages.

b) delay - the mean and maximum delay that the protocol will add to process responsiveness during the peak period.

c) cost - maximum acceptable recurring and nonrecurring costs associated with the installation of a protocol in a local area network.

A protocol may perform functions at the communication link level or at application process level. Of primary concern to local area network application developers are the application level protocols. Some of the protocols which might be considered elementary to local area network developments are discussed below.

2. Application subsystem perspective of elementary protocols

The need for a number of elementary high-level protocols providing various types of services is apparent. Three main groupings of such protocols may be defined: application-oriented, executive-oriented, and network-induced. The motivation for this classification is the perspective of distributed systems as extensions of the single-system environment. The goal of elementary protocols is to extend the array of system utilities, programs and operating system services that are available on a single system to the total local area network. Hence, development of elementary local area network protocols is a basic step in evolving high-level operating systems for local area networks.

a. Application-Oriented Protocols

Application-oriented protocols are defined to be interprocess communication rules and data formats which extend the commonly used system programs (languages,

editors, library, etc.) of one node to an application process in another node. Such protocols may include:

1). File Transfer - allowing a process in one node to access files on another node as though the process were executing in that node. Similar to general system utilities that support media conversion with and without blocking changes, format changes, naming changes, etcetera.

2). Editor - allowing a process in one node to store, modify and retrieve text information in a file at another node. Preferably the protocol is implemented to a specification consistently applied at each node on the network.

3). Compile - allowing a process in one node to produce executable programs at another node. The protocol implements a network wide source and object code library and linkage editor.

4). Execute - allowing a process in one node to invoke a program at another node by module name, to supply parameters to the program and to receive program output and system notice messages at the sending local area network node process.

5). Debug - allowing a process in one node to dynamically debug a program at another node. Preferably allows an application process distributed among two or more local area network nodes to be debugged interactively.

b. Executive-Oriented Protocols

Executive level protocols are defined to be interprocess communication rules and data formats which extend the operating system services (resource allocation,

device or program service, monitor services, etc.) of one node to an application process in another node.

1). Command Protocol - allowing commonly used operating system services (ASSIGN, PRINT, TIME, STATUS, etc.) in each node of a network to be invoked uniformly by a process in another node.

2). Virtual Scrolling Terminal Protocol - allowing a process in one node to communicate with a process in another node as though the receiving process were a scrolling output device such as a printer or teletype.

3). Virtual Screen Terminal Protocol - allowing a process in one node to communicate with a process in another node where the receiving process operates on a randomly addressable collection of two dimensional pages of text using predefined functions and transmitted variables.

4). Virtual Graphic Terminal Protocol - allowing a process in one node to communicate with a process in another node where the receiving process operates on a randomly addressable collection of two or three dimensional figures and two dimensional text using predefined functions and transmitted variables.

c. Network-Induced Protocols

Network induced protocols are defined to be interprocess communication rules and data formats which facilitate the operation of executive level and application level protocols in a local area network.

1). Network Endpoint Declaration Protocol - provides the mechanism for a local area network node to establish or disestablish addressable network ports in a directory thereby allowing qualified processes in other nodes to

become associated with processes assigned to this port. The protocol might serve normal and privileged processes in the application space as well as network control functions within the operating system. This protocol provides a mechanism to identify "well known" processes in a network directory.

2). Network Access Authorization Protocol - allowing a process to gain access to another process in the network. Includes log-on/log-off support to end users as well as general process interconnection authorization. Interfaced with network security and privacy management information systems.

3). Network Directory Service Protocol - allowing a process to request information about a node, another process or an end user. May also support custom menu services for each network user to promote the impression of a single integrated system.

4). Transport Control Protocol - allowing a process in one node to establish an association with a process in another node and to exchange messages in a virtual circuit or datagram mode. Usually implemented as an augmentation to the operating system of a network node.

5). Interprocess Synchronization - providing a mechanism for two or more processes in two or more nodes to coordinate asynchronously executing functions. This protocol could underlie the virtual terminal control protocol.

6). Network System Control Protocol - providing the mechanism for establishing "built-in" maintenance and security subsystems in a local area network environment. It is envisioned that performance, maintenance

and security checks should permeate the local area network software as well as hardware system. This protocol facilitates unified specification of performance, maintenance and security related functions.

B. PROTOCOL HIERARCHIES

To reduce their design complexity, most networks are organized as a series of layers or levels as mentioned earlier.

Layer n on one machine carries on a conversation with layer n on another machine. The rules and conventions used in this conversation are collectively known as the layer n protocol, as illustrated in Fig. 2.2 for a seven-layer network. The entities comprising the corresponding layers on different machines are called peer processes. In other words, it is the peer processes that communicate using protocol.

In reality, no data are directly transferred from layer n on one machine to layer n on another machine (except in the lowest layer). Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. At the lowest layer there is physical communication with the other machine, as opposed to the virtual communication used by the highest layers. In Fig. 2.2 virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers there is an interface. The interface defines which primitive operations and services the lower layer offers to the upper one. When network designers decide how many layers to include in a network and what each one should do, one of the most important considerations is having cleanly defined intefaces between the layers. Having cleanly defined interfaces, in

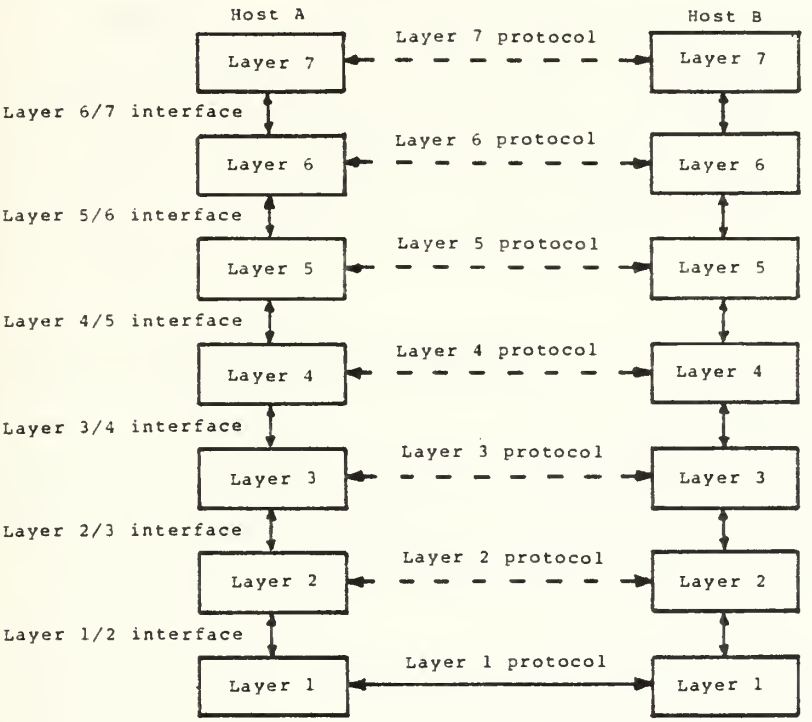


Figure 2.2 Layers, Protocols, and Interfaces.

turn, requires that each layer perform a specific collection of well understood functions. In addition to minimizing the amount of information that must be passed between layers, clean cut interfaces also make it simpler to replace the implementation of one layer with a completely different one, because all that is required of the new implementation is that it offers exactly the same set of services to its upstairs neighbor as the old implementation did.

The set of layers and protocols is called the network architecture [Ref. 2.]. The specification of the architecture must contain enough information to allow an implementer to write the program for each layer so that the program will correctly obey the appropriate protocol. Neither the details of the implementation nor the specification of the interfaces are part of the architecture. In fact, it is not even necessary that the interfaces on all machines in a network be the same, provided that each machine can correctly use all the protocols.

C. DESIGN ISSUES FOR THE LAYERS

Some of the key design issues that occur in computer networking are present in several layers. The following are some of the common problems that must be repeatedly dealt with in the design of the different protocols.

1. Every layer must have a mechanism for connection establishment. Since a network normally has many computers, some of which have multiple processors, some means is needed for a process on one machine to specify who it wants to talk to. In any layer where there are multiple destinations, addressing is needed.

2. Closely related to the mechanism for establishing connections across the network is the mechanism for terminating them once they are no longer needed.

3. Another set of design decisions are the rules for data transfer. Does data only travel in one direction, called simplex communication, or can data travel in either direction, but not simultaneously, called half-duplex communication, or can they travel in both directions at once, called full-duplex communication? The protocol must also determine how many logical channels the

connection corresponds to, and what their priorities are. Many networks provide at least two logical channels per connection, one for normal data and one for urgent data.

4. Error control is an important issue when the physical communication circuits are not perfect. Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.

5. Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol make explicit provision for the receiver to allow the pieces to be put back together properly.

6. An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. There are various solutions to this and all of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about what the receiver's current situation is.

7. Another problem that must be solved repeatedly at different levels is the inability of all processes to accept arbitrarily long messages. This leads to mechanisms for disassembling, transmitting, and then reassembling messages.

D. THE ISO REFERENCE MODEL

This model is closely based on a proposal developed by the International Standard Organization (ISO) as a first step toward international standardization of the various protocols.

The Reference Model of Open System Interconnection has seven layers as shown in Fig. 2.3.

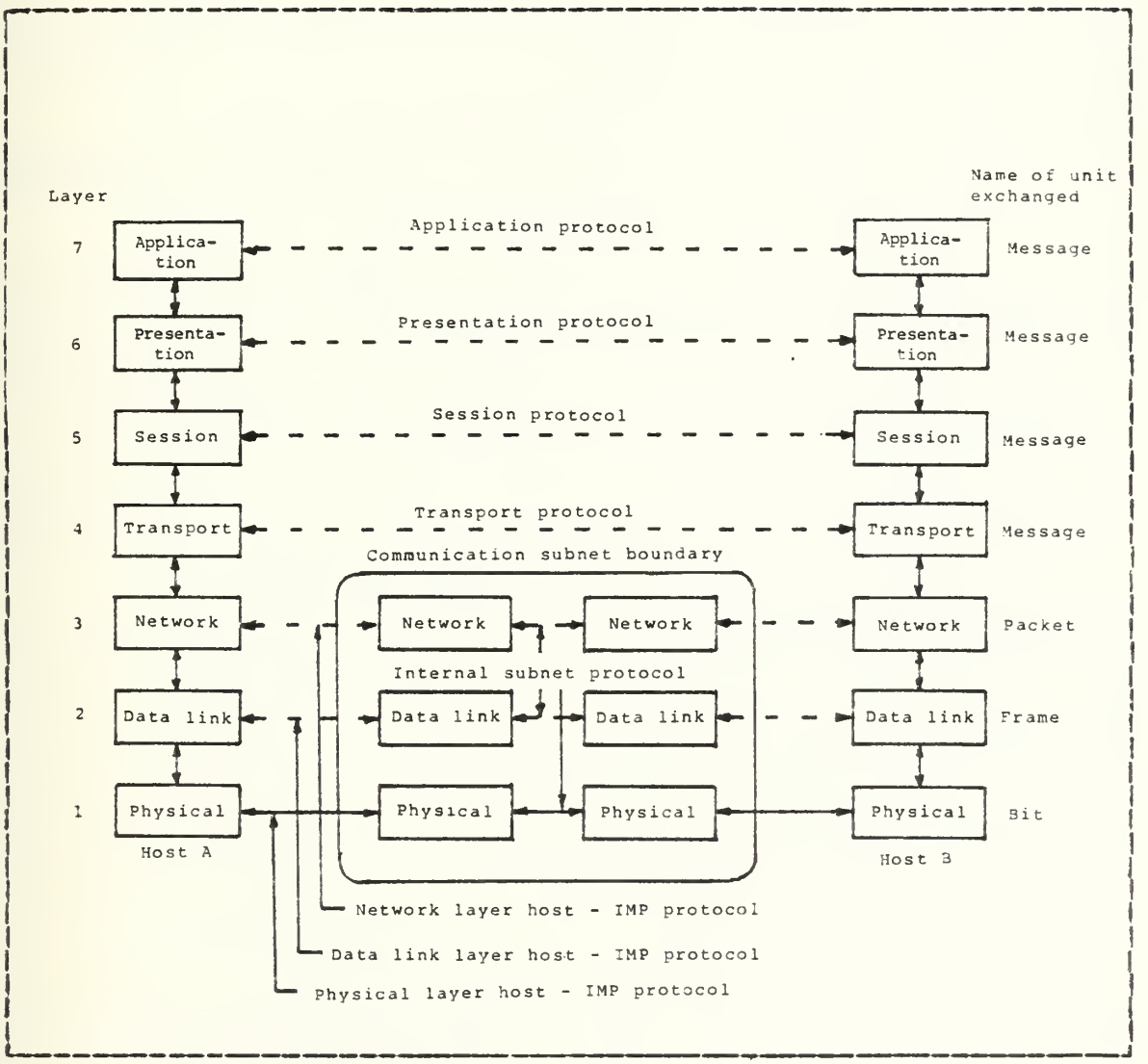


Figure 2.3 ISO Reference Model.

The principles that ISO applied to arrive at the seven layers are as follows:

1. A layer should be created where a different level of abstraction is needed.

2. Each layer should perform a well defined function.

3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.

4. The layer boundaries should be chosen to minimize the imformation flow across the interfaces.

5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

The seven layers, from the lowest layer to the highest layer, are:

1.Physical layer - The physical layer provides mechanical, electrical, functional and procedural characteristics to establish, maintain, and release physical connections (e.g., data-circuits) between link-entities. The physical layer provides for the transmission of transparent bit streams between data link layer protocols across physical connections which are permanently or dynamically established.

2.Data Link Layer - The purpose of the data link layer is to provide the functional and procedural means to establish, maintain, and release one or more data links among network-entities. This layer

masks the characteristics of the physical layer (such as switched, multipoint, broadcast, polling, contention, etc.) from the network layer.

3. Network Layer - The network layer provides functional and procedural means to exchange network-services-data-units between two transport-entities over a network-connection. It provides transport-entities with independence from routing and switching considerations, including the case where a tandem subnetwork-connection is used. The network layer protocol uses underlying data link connections to make network connections invisible to the transport layer protocol.

4. Transport Layer - The transport layer exists to provide a universal transport service in association with the underlying services provided by lower layers. The transport-service provides transparent transfer of data between session-entities. The transport-service relieves these session-entities from any concern with the detailed way in which reliable and cost-effective transfer of data is achieved. Three types of transport services are:

- A connection-oriented service
- A transaction-oriented service
- A broadcast-oriented service

The transport service is required to optimize the use of the available communications services to provide the performance required for each connection between session-entities at a minimum cost. To achieve optimization, the global demands of all concurrent transport users and the transport layer resource limitations are considered.

5.Session Layer - The purpose of the session layer is to assist in the support of the interactions between cocrperating presentation-entities. To do this, the session layer provides services which are classified into the following two categories:

a) Session Administration Services - binding two presentation-entities into a relationship and unbinding them.

b) Session Dialogue Service - control of data exchange, delimit and synchronizing data operations between two presentation-entities.

6.Presentation Layer - The purpose of the presentation layer is to provide the set of services which may be selected by the application layer to enable it to interpret the meaning of the data exchanged. These services are for the management of the entry, exchange, display and control of structured data. The presentation-service is location independent and is considered to be on tcp of the session layer which provides the service of linking a pair of presentation-entities. It is through the use of services provided by the presentation layer that applications in an open systems interconnection environment can communicate without unacceptable costs in interface variability, transformations or application modification. There are four phases of presentation layer protocol operation:

a) Session establishment phase in which the connection is set up.

b) Presentation image control phase in which the presentation options can be selected by value, by name, by prior agreement or by negotiation.

c) Data transfer phase which controls the data structure accesses and perhaps executes special purpose transformations such as voice compression or data encryption.

d) Termination phase

7. Application Layer - This is the highest layer in the reference model of open systems interconnection architecture. Protocols of this layer directly serve the end user by providing the distributed information service appropriate to an application, to its management and to the system management. Management of open systems interconnection comprises those functions required to initiate, maintain, terminate and record data concerning the establishment of connections for data transfer among application processes. The other layers exist only to support this layer. Three categories of application layer protocols are defined:

a) System Management Protocols - responsible for controlling and supervising open systems (e.g., initiating dialog).

b) Application Management Protocols - responsible for controlling and supervising application processes (e.g., access control).

c) System Protocols - responsible for executing information processing functions on behalf of an application process or user (e.g., electronic mail).

III. ETHERNET LOCAL AREA NETWORK

A. OVERVIEW

Ethernet is one type of local communication network which makes use of coaxial cable as a mean to transfer data.

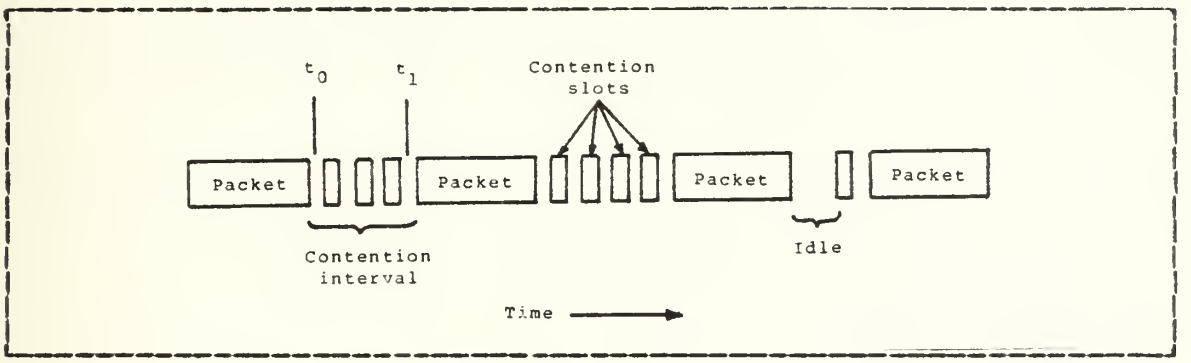


Figure 3.1 Contention, Transmission, and Idle States.

All stations in the Ethernet network monitor the cable (the ether) during their own transmission, terminating transmission immediately if a collision is detected.

The Ethernet mechanism is modeled in Fig. 3.1. At the point marked t_0 a station has finished transmitting its packet. Any other stations having a packet to send may now attempt to do so. If two or more stations decide to transmit simultaneously, there will be a collision. Each will detect the collision, abort its transmission, wait a random period of time, and then try again, assuming that no other station has started transmitting in the mean time. Ethernet will therefore consist of alternating contention and transmission periods, with idle periods occurring when all stations are quiet.

B. CONCISE ETHERNET SPECIFICATION

1. Packet Format

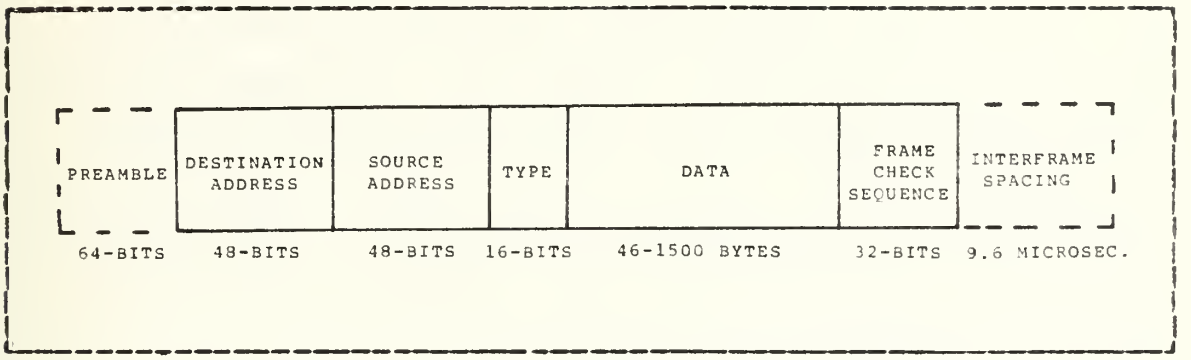


Figure 3.2 Ethernet Packet Format.

A station must be able to transmit and receive packets on the common coaxial cable with the indicated packet format and spacing. Each packet should be viewed as a sequence of 8-bit bytes; the least significant bit of each byte (starting with the preamble) is transmitted first.

a). Maximum Packet Size: 1526 bytes(8 byte preamble + 14 byte header + 1500 data bytes + 4 byte CRC)

b). Minimum Packet Size: 72 bytes (8 byte preamble + 14 byte header + 46 data bytes + 4 byte CRC)

c). Preamble: This 64-bit synchronization pattern contains alternating 1's and 0's, ending with two consecutive 1's.

d). Destination Address: The 48-bit field specifies the station(s) to which the packet is being transmitted. Each station examines this field to determine whether it should accept the packet. The first bit transmitted

indicates the type of address. If it is a 0, the field contains the unique address of the one destination station. If it is a 1, the field specifies a logical group of recipients; a special case is the broadcast (all stations) address, which has all 1's.

e). Source Address: This 48-bit field contains the unique address of the station that is transmitting the packet.

f). Type Field: This 16-bit field is used to identify the higher level protocol type associated with the packet. It determines how data field is interpreted.

g). Data Field: The field contains an integral number of bytes ranging from 46 to 1500. (The minimum ensures that valid packets will be distinguishable from collision fragments.)

h). Packet Check Sequence: This 32-bit field contains a redundancy check (CRC) code, defined by the generating polynomial:

$$G(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11} +x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$$

The CRC covers the address (destination/source), type, and data fields. The first transmitted bit of the destination field is the high-order term of the message polynomial to be divided by G(x) producing remainder R(x). The high-order term of R(x) is the first transmitted bit of the Packet Check Sequence field. The algorithm uses a linear feedback register which is initially preset to all 1's. After the last data bit is transmitted, the contents of this register (the remainder) are inverted and transmitted as the CRC field. After receiving a good packet, the receiver's shift register contains 11000111 00000100 11011101 01111011 (x^{31}... ,x^{0}).

i). Minimum Packet Spacing: This spacing is 9.6 microsecond, The minimum time that must elapse after one transmission before another transmission may begin.

j). Round-Trip Delay: The maximum end-to-end, round-trip delay for a bit is 51.2 microsecond.

k). Collision Filtering: Any received bit sequence smaller than the minimum valid packet (with minimum data field) is discarded as a collision fragment.

2. Control Procedure

The control procedure defines how and when a host station may trasmit packets into the common cable. The key purpose is a fair resolution of occasional contention among transmitting stations.

a). Defer: A station must not transmit into the coax cable when the carrier is present or within the minimum packet spacing time after the carrier has ended.

b). Transmit: A station may transmit if it is not deferring. It may continue to transmit until either the end of the packet is reached or a collision is detected.

c). Abort: If a collision is detected, transmission of the packet must terminate, and a jam (4-6 bytes of arbitrary data) is transmitted to ensure that all other participants in the collision also recognize its occurrence.

d). Retransmit: After a station has detected a collision and aborted, it must wait for a random retransmission delay, defer as usual, and then attempt to retransmit the packet. The random time interval is computed using the backoff algorithm (below). After 16 retransmission attempts, a higher level (e.g. software) decision is

made tc determine whether to continue or abandon the effort.

e). Backoff: Retransmission delays are computed using the Truncated Binary Exponential Backoff algorithm, with the aim of fairly resolving contention among up to 1024 stations. The delay (the number of time units) before the nth attempt is a uniformly distributed random number from 0 to 2 for $0<n\leq10$ (n = 0 is the original attempt). For attempt 11-15, the interval is truncated and remains at 0 to 1023. The unit of time for the retransmission delay is 512 bit times (51.2 microsecond).

3. Channel Encoding

Manchester encoding is used on the coaxial cable. It has a 50% duty cycle, and insures a transition in the middle

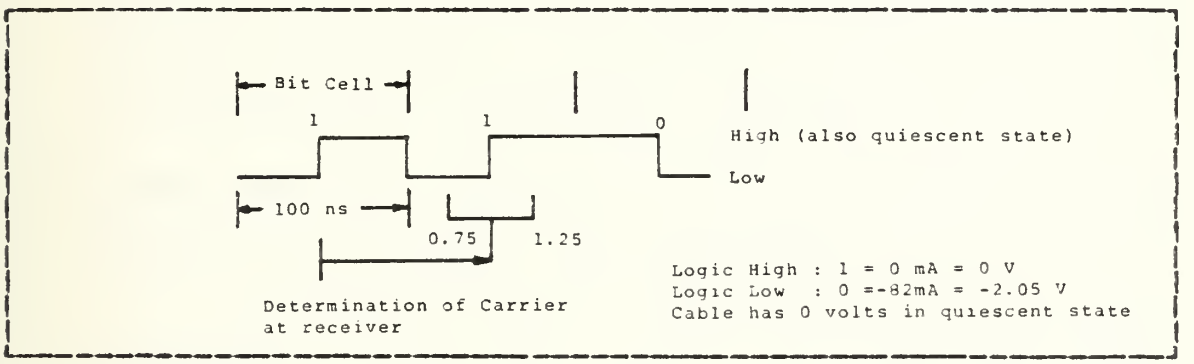


Figure 3.3 Data Rate Scheme.

of every bit cell ("data transition"). The first half of the bit cell contains the complement of the bit value, and the second half contains the true value of the bit.

4. Data Rate

Data rate is 10 Mbit/sec = 100 nsec bit cell ± 0.01%.

5. Carrier

The presence of data transitions indicates that carrier is present. If a transition is not seen between 0.75 and 1.25 bit times since the center of the last bit cell, then carrier has been lost, indicating the end of a packet. For purposes of deferring, carrier means any activity on the cable, independent of being properly formed. Specifically, it is any activity on either receive or collision detect signals in the last 160 nsec.

6. Coax Cable

a). Impedance: 50 ohms ± 2 ohms (Mil Std. C17-E). This impedance variation includes batch-to-batch variations. Periodic variations in impedance of upto ± 3 ohms are permitted along a single piece of cable.

b). Cable Loss: The maximum loss from one end of a cable segment to the other end is 8.5 db at 10 MHz (equivalent to 500 meters of low loss cable).

c). Shielding: The physical channel hardware must operate in an ambient field of 2 volts per meter from 10 MHz to 30 MHz and 5 volts per meter from 30 MHz to 1 GHz. The shield has a transfer impedance of less than 1 millichm per meter over the frequency range of 0.1 MHz to 20 MHz (exact value is a function of frequency).

d). Ground Connection: The coax cable shield shall not be connected to any building or AC ground along its length. If for safety reasons a ground connection of the shield is necessary, it must be in only one place.

e). Physical Dimensions: This specifies the dimension of a cable which can be used in the standard tap. Other cables may also be used, if they are not to be used with a tap-type transceiver (such as used with connectorized transceivers, or as a section between sections to which standard taps are connected).

Center Conductor:	0.0855" diameter solid tinned copper
Core Material:	Foam polyethylene or foam teflon FEP
Core O.D.:	0.242" minimum
Shield:	0.326" maximum shield O.D.
Jacket:	PVC or teflon FEP
Jacket O.D.:	0.405"

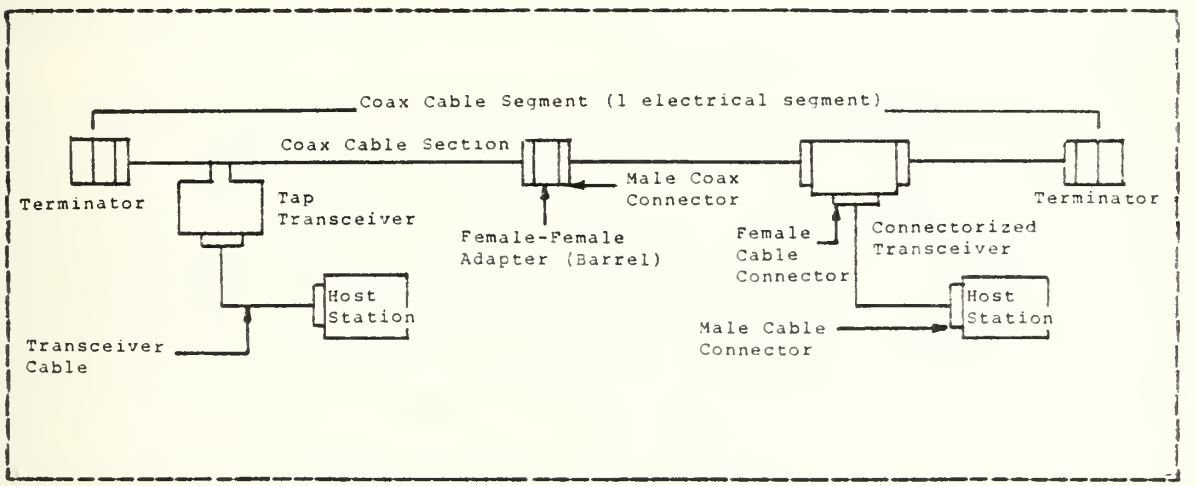


Figure 3.4 Coax Cable, Connectors, and Transceivers.

7. Coax Connectors and Terminators

Coax cables must be terminated with male N-series connectors, and cable sections will be joined with female-female adapters. Connector shells shall be insulated such that the coax shield is protected from contact to building

grounds. A sleeve or boot is acceptable. Cable segments should be terminated with a female N-series connector (can be made up of a barrel connector and a male terminator) having an impedance of 50 ohms + 1%, and able to dissipate 1 watt. The outside surface of the terminator should also be insulated.

8. Transceiver

Up to 100 transceivers may be placed on a cable segment no closer than 2.5 meters. Following this placement rule reduces to a very low (but not zero) probability the chance that objectionable standing waves will result. The details of transceiver interface and coax cable interface can be found in Interlan's "Concise Ethernet Specification" [Ref. 3].

C. NI1010 UNIBUS ETHERNET COMMUNICATIONS CONTROLLER

1. Features

a). Ethernet Data Link Layer Functions:
   1). Data Encapsulation/decapsulation
   2). Carrier Sense Multiple Access/Collision Detected (CSMA/CD) transmit and receive data link management

b). Ethernet Physical Channel Functions:
   1). 10 Mbits per second data rate
   2). Data encoding and decoding
   3). Channel access
   4). Transceiver cable interface

c). Network Statistics:
   1). Tallies number of transmissions, receptions, errors, and collisions

d). Performance:

1).16 Kbytes FIFC buffer for back-to-back frame reception

2).2 Kbyte FIFO buffer for frame transmission

3).DMA transfers to/from unibus memory

e).Extensive Diagnostic Features:

1).internal and external data loop-back operation

2).Network LED indicators

3).Pcwer-up confidence test

4).Pass/fail LED indicator

5).Diagnostic software provided

f).Cne Hex-Hight Board:

1).Fit one unibus SPC slot

2. Description

The NI1010 Unibus Ethernet Communication Controller Board is a single Hex-height board that contains all the data communications ccntroller logic required for interfacing DEC's family of VAX-11 and Unibus-based PDP-11 minicomputers tc the Ethernet local area network. It performs the specified data link and physical channel functions, permitting Unibus-based systems to engage in transmission and reception of data with other Ethernet stations on the local area network with the speed of 10 Mbit per second and with the maximum distance of 2500 meters. As shown in Figure 3.5, the NI1010, when attached to a transceiver unit, provides a VAX-11 or Unibus-based PDP-11 a complete connection cntc the Ethernet local area network.

3. Ethernet data link layer functions

Within the data link layer the NI1010 performs the specified Ethernet transmitter processes of Transmit Data Encapsulation and Transmit Link Management, and the Ethernet receive processes of Receive Data Decapsulation and Receive Link Management.

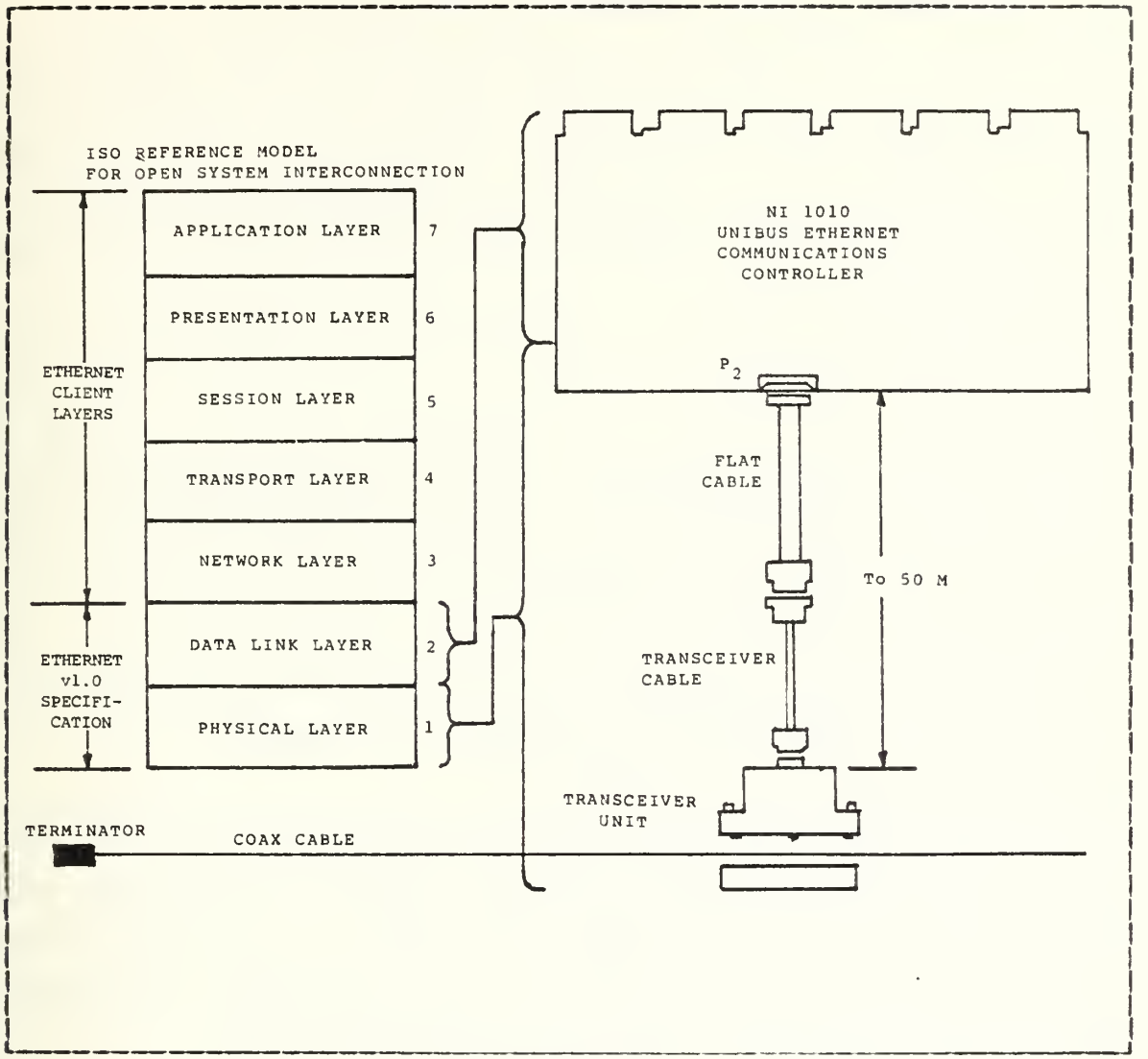


Figure 3.5 Ethernet Architecture and Implementation.

a. Transmit data encapsulation

Figure 3.6 shows the Ethernet Frame Format for packet transmissions over the coaxial cable physical channel. For receive synchronization purposes, the frame is preceeded with a 64-bit preamble sequence and terminated with a minimum interframe spacing period of 9.6 microseconds.

The Destination Address field specifies the station(s) for which the frame is intended. The address value provided by the user may be either; 1) the physical address of a particular station on the network; 2) a multicast-group address associated with one or more stations; or 3) the broadcast address for simultaneous tran-

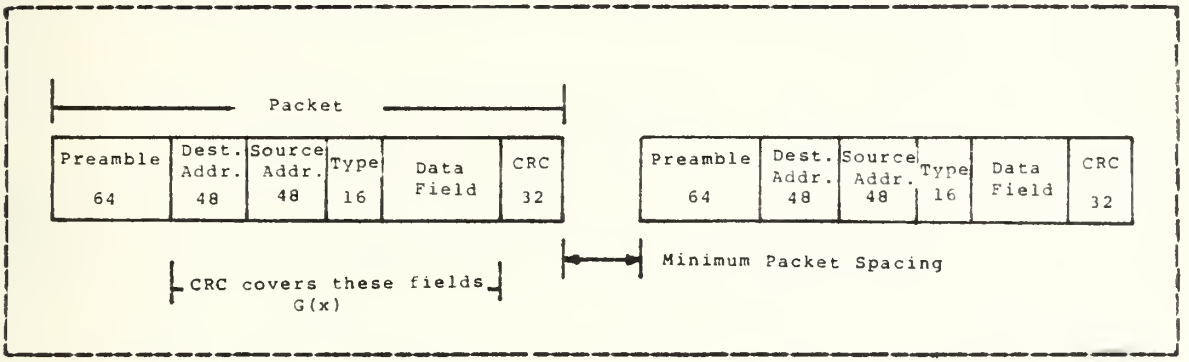


Figure 3.6 Ethernet Frame Format.

smission to all stations on the network. The first bit of the Destination Address distinguishes a physical address from a multicast address (0 = physical, 1 = multicast). For broadcast transmissions an all one-bit pattern is used.

The Source Address field specifies the physical address of the transmitting station. To eliminate the possibility of an addressing ambiguity on a network, associated with each NI1010 is a unique 48-bit physical address value assigned to it at the time of manufacture. On transmission, the NI1010 inserts this value into the Source Address field.

The type field is specified by the user for use by high level network protocols. It specifies to the receiving station(s) how the content of the Data field is to be interpreted.

The Data field may contain a variable number of data bytes ranging from a minimum of 46 bytes to a maximum of 1500 bytes. The NI1010 accepts less than 46 bytes from the user by automatically inserting null characters to complete a 46-byte minimum frame size.

The Frame Check Sequence (FCS) field contains a 32-bit cyclic redundancy check (CRC) value generated by the NI1010 during transmission.

b. Transmit link management

The NI1010 performs all Ethernet Transmit Link Management functions required to successfully deliver a frame onto the network. These functions include:

- Carrier Deference; the NI1010 monitors the physical channel and defers its transmission should the channel be busy carrying other traffic;
- Collision Detection; once the NI1010 has finished deferring to the passing traffic on the network, it proceeds with its own transmission. In the event that another station simultaneously began a transmission, a "collision" occurs. The NI1010 detects this event and terminates its transmission attempt; and
- Collision Backoff and Retransmission; when a transmission attempt has been terminated due to a collision the NI1010 attempts its transmission again after delaying a short random period of time. The scheduling of the retransmission is determined by the Ethernet process called "truncated binary exponential backoff". The NI1010 reports an error should it be unable to deliver its frame onto the network after 16 transmission attempts.

c. Receive data decapsulation

When not transmitting a frame the NI1010 continuously listens to the traffic being carried on the network. After synchronizing to the preamble sequence of a frame on the network, the NI1010 processes the Destination Address field through its address filter logic to determine whether or not the incoming frame is intended for it. The NI1010 controller will only accept a frame from the network with a Destination Address value that either:

1) matches the physical address of the NI1010 board itself;
2) contains the broadcast address; or
3) matches one of the 63 multicast-group logical addresses which the user may assign to the board.

The NI1010 performs high speed multicast-group address recognition. Whenever a multicast-group logical address is received on the network, the NI1010 converts the frame's 48-bit Destination Address field into a 6-bit table entry pointer through the application of a many-to-few mapping called "hashing". It uses the resulting pointer to look into a table of valid multicast-group addresses to see if the received address is one that the station should accept.

For network management and diagnosis, the NI1010 may be operated in a "promiscuous" receive mode. When in this mode, the NI1010 disables its address filter logic and accepts all undamaged frames passing on the network.

The NI1010 validates the integrity of a received frame by regenerating the 32-bit CRC value on the received bit stream and comparing it against the CRC value found in the frame's Frame Check Sequence field.

d. Receive link management

Since collisions are a normal occurrence in the Ethernet's CSMA/CD link management process, the NI1010 receiver filters out collision fragments from valid frames.

4. Ethernet physical layer functions

Within the Ethernet Physical Layer the NI1010 performs the electrical and procedural specifications required for interfacing directly to a transceiver unit. Transmissions and receptions take place at a 10 Mbits per second data rate under half-duplex operation.

a. During transmission the NI1010's physical channel functions include:

1). Generating the 64-bit preamble sequence for all receivers on the network to synchronize on;
2). Parallel to serial conversion of the frame;
3). Calculating a 32-bit CRC value and inserting it into the Frame Check Sequence field;
4). Generating a self-synchronizing serial bit stream through Manchester encoding of the data; and
5). Providing proper channel access by detecting carrier from another station's frame transmission, and sensing the collision presence signal from the transceiver unit.

b. The NI1010's physical channel functions during reception include:

1). Manchester decoding the incoming bit stream into a data stream and a clock stream;

2). Synchronizing to, and removal of, the preamble sequence; and
3). Serial to Parallel conversion of the frame.

5. Performance

The NI1010 has been designed to offer high network performance while minimizing the service loads placed upon the host Unibus system.

Serving to buffer the system from the unpredictable interarrival times characteristic of network traffic, the board has a FIFO (first-in, first-out) memory which can store up to 16 Kbytes of received frames. Because of this extensive front-end buffering, few time-critical service requirements are imposed on the host Unibus system.

For transmission, the NI1010 has a 2 Kbyte Transmit FIFO which permits the host to perform a one-time transfer of a frame.

All data block transfers between the NI1010 and Unibus memory are performed under the control of an onboard DMA controller. To maximize system performance during reception, the controller allows the user to preload up to sixteen different memory buffer address and byte count values for DMA of received frames.

6. Extensive Diagnostic Features

The NI1010 offers comprehensive network and board-level diagnostic tools which greatly simplify the process of identifying a network communication problem. Mounted on the edge of the board are four network state LED indicators which provide a visual indication of whether or not the user's station is communicating onto the network. For a comprehensive station diagnosis, the user can exercise the NI1010's communication facilities in either internal or

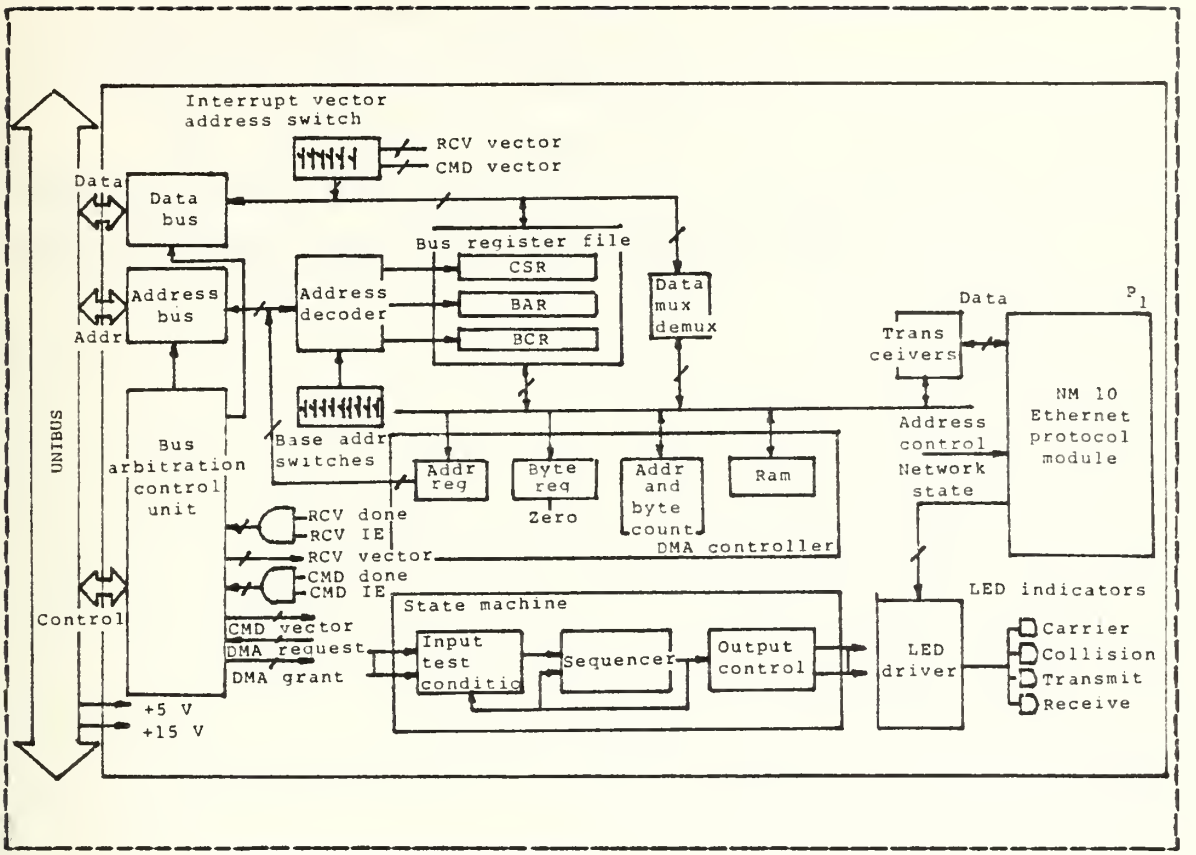


Figure 3.7 NI1010 Functional Diagram.

external data loopback mode; making it possible to detect and isolate a fault to the coaxial cable, transceiver unit, transceiver cable; or the NI1010 board itself.

On power-up the NI1010 performs a confidence test of the onboard memories, register and data paths. A LED indicator shows the pass/fail operational state of the board.

7. Network statistics

The NI1010 collects network statistics to permit the user to characterize network operation. Statistics tallied include:

a). number of frames received

b). number of frames received with CRC error

c). number of frames received with alignment error

d).number of frames transmitted

e).number of transmit collisions

For detailed description of NI1010 Ethernet Communication Controller Board, see NI1010A Unibus Ethernet Communications Controller User Manual [Ref. 4].

IV. DETAILED DESIGN AND IMPLEMENTATION

A. DESIGN ISSUES CONSIDERATION

After studying the NI1010 Unibus Ethernet Communications Controller in detail, some of the common problems with the design of protocol mentioned in Chapter II can be solved as follows:

1. The method needed for a process on one machine to specify who it wants to talk to is the Destination Address and the Source Address field in every transmitted frame.

2. There is no need to find a mechanism for terminating the connection across the network, since the Ethernet will put itself to the idle state automatically if there is no carrier on the coax cable.

3. The rule for data transfer is fixed on the half-duplex operation, i.e., the data can travel in either direction, but not simultaneously.

4. For the error control issue, Ethernet frame format provides a 32-bit Frame Check Sequence field which contains Cyclic Redundancy Check (CRC) value. This value is generated by the Ethernet board of the sending station and will be checked by the board of the receiving station.

5. The problem of how to keep a fast sender from swamping a slow receiver with data is solved by the 16 Kbyte FIFO Receive buffer on all Ethernet boards.

6. Ethernet board performs data link layer in transmitting data encapsulation and receive data decapsulation. The data field of the frame format can vary from 46 to 1500 bytes. Thus, the design issue of the ability of processing arbitrarily long messages is solved. Furthermore, if the data is less than 46 bytes, the board will automatically insert null characters to complete a 46-byte minimum frame size.

However, the problem of preserving the order of messages is not solved by the capability of the Ethernet board. The design of a higher layer protocol would have to take this issue into consideration. The forthcoming sections will explain how to create a design such that it will overcome this problem.

The NI1010 Unibus Ethernet Communications Controller Board together with the Transceiver represent the first two layers, Physical Layer and Data Link Layer, of the ISO Reference Model as shown in Figure 4.1.

B. NS2030 DEVICE DRIVER

1. Overview

The NS2030 product includes a diagnostic program and a VAX/VMS device driver source, hereafter referred to as NIDRIVER, that allows a suitably privileged application program written in VAX-11 MACRO assembly language or any VAX/VMS high level language (such as VAX-11 FORTRAN) to interface to INTERLAN's Unibus to Ethernet interface, the NI1010. The application program uses standard VAX/VMS services to interface to the Ethernet. There are no new interfaces to learn. NIDRIVER supports all features of the NI1010 controller, including full duplex operation (transmitting with receives outstanding), non-contiguous buffers

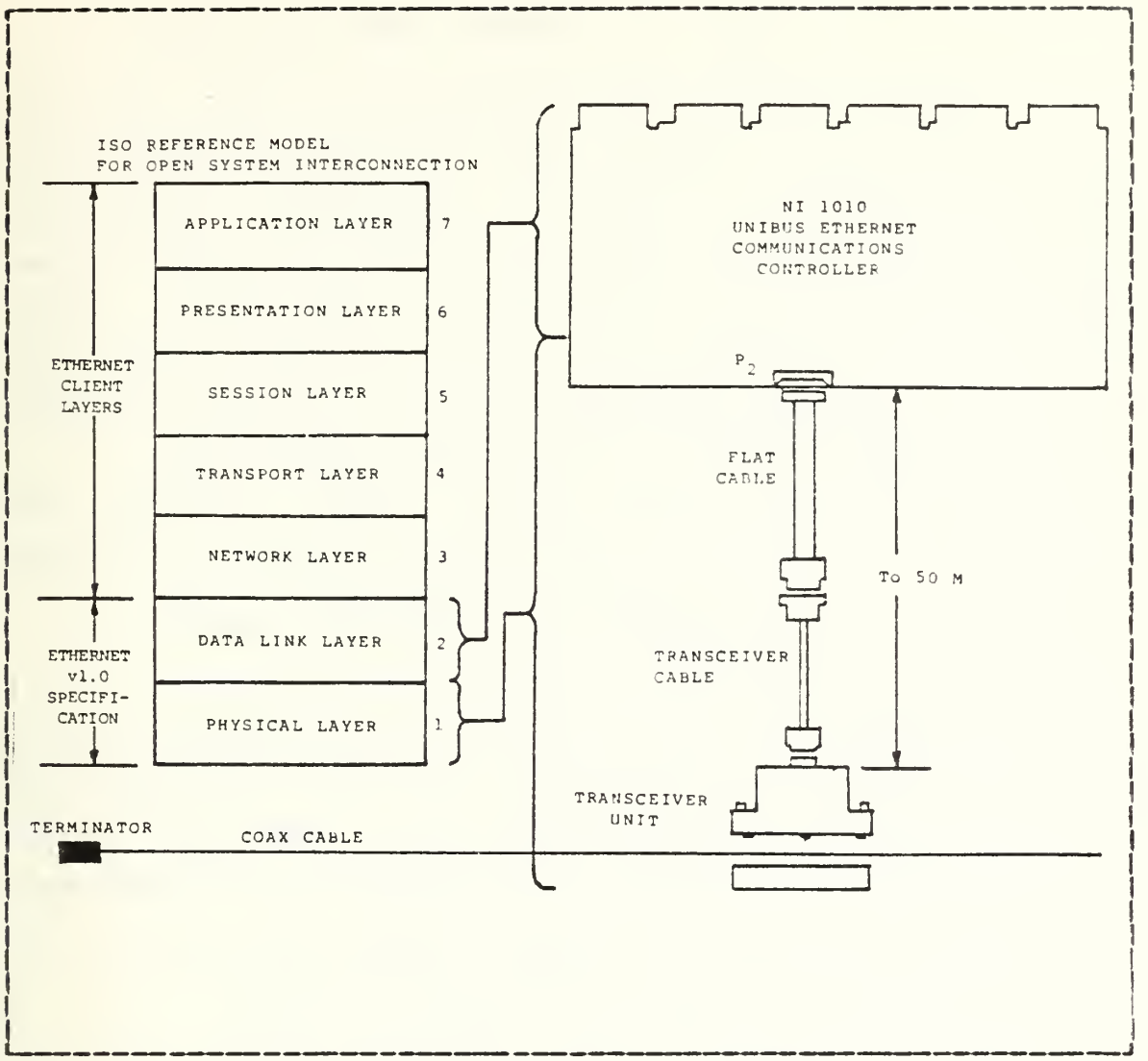


Figure 4.1 NI1010 and ISO Reference Model.

for transmission and reception (often called "scatter/gather" or "buffer chaining"), and the extensive onboard intelligence and diagnostic functions. The standard QIO interface allows the network software designer to choose among several techniques for performing I/O: synchronous I/O using the $QIOW system service, and asynchronous I/O using the $QIO system service with event flags and/or AST routines.

With one $QIO request, an application program is able to instruct the NI1010 to obtain a preformatted Ethernet packet out of system memory and transmit it onto the Ethernet. Similarly, with one $QIO request, the program can specify the address and length of a buffer in system memory into which the controller can place the next received packet. Single $QIO requests can also be used to perform other NI1010 functions such as GO ON-LINE, RUN DIAGNOSTICS, REPORT STATISTICS, and LOAD GROUP ADDRESS(ES).

The detailed description of NIDRIVER is contained in the Interlan NS2030 VAX/VMS (TM) Device Driver and Diagnostics User Manual [Ref. 5].

2. Program interfaces to NIDRIVER

Detailed descriptions of the following standard VAX/VMS system services can be found in the VAX/VMS System Services Reference Manual [Ref. 6] and in the VAX/VMS I/O User's Guide [Ref. 7]. Some of the very important system services are also included in Appendix A.

a. Using $assign (associate channel) with NIDRIVER

Before a program can issue requests to NIDRIVER, it must assign a channel to the NI1010 controller. The Assign I/O Channel ($ASSIGN) system service is used to assign a channel to a device. You supply the device name as part of the $ASSIGN call: $ASSIGN returns a channel number. The NI1010 controller device name supported by NIDRIVER is of the form NIxy: Because the NI1010 controller represents a single "unit" (in the VMS I/O sense), the first controller is called "NIA0:", the second "NIB0:", and so on.

b. Using $ALLOC (allocate device) with NIDRIVER

A process can allocate a device for its exclusive use using the $ALLOC system service. Once the device is allocated, no other process (except for subprocesses related to the issuing process) can assign a channel to the device.

Because the NI1010 controllers provide low level access to the Ethernet. NIDRIVER supports each controller as a non-sharable device. When a process assigns a channel to an NI1010 controller, VAX/VMS performs an implicit $ALLOC for the process.

c. Using $GETCHN and $GETDEV (get device Information)

Two system services can be used to obtain information about NIDRIVER: Get Channel Information ($GETCHN) and Get Device Information ($GETDEV). $GETCHN is used to obtain information about a specific device.

When used to obtain information about an NI1010 controller, these system services return identical primary and secondary device characteristics.

d. Using $QIO and $QIOW (request I/O function)

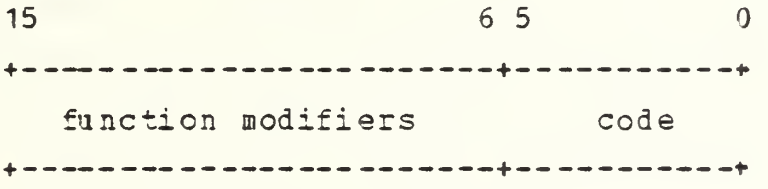
Because NIDRIVER supports the standard VAX/VMS QIO interface, all controller requests follow the general QIO format:

```
$QIO_S  [efn],chan,func,[iosb],[astadr],[astprm],
        [p1],[p2],[p3],[p4],[p5],[p6]
```

The first six arguments are device/function independent and can be used in any controller I/O request. For example, you can specify an AST routine address (the "astadr" argument of the QIO request) if you need to execute special code at I/O completion.

Device/function dependent arguments P1 and P2 must be supplied for all controller operation that use the DMA channel for transferring data to or from VAX memory. P1 is the (virtual) address of a WORD-ALIGNED buffer. P2 is the size of the buffer in bytes and must be even and less than 1536 (decimal). Parameter P3 through P6 are ignored in NI1010 operations.

(1). I/O functions. To fully understand the I/O functions supported by NIDRIVER, one should know how VAX/VMS I/O functions are encoded into 16-bit values (the function argument of the QIO request). I/O function values have the following format:



The low-order 6 bits of the function value are a code that specifies the particular operation to be performed. The high-order 10 bits of the function value are function modifiers and are normally used to alter the particular operation specified by the code. Symbolic names for function codes and modifiers are defined by the $IODEF macro, as described in the VAX/VMS System Services Reference Manual [Ref. 6]. A modified function can be invoked by "OR"ing a function code and function modifier. For example:

```
IO$_SETMODE!IO$_SHUTDOWN
```

in MACRO assembly language, or

```
IO$_SETMODE .OR. IO$_SHUTDOWN
```

in FORTRAN.

The following describes each I/O function supported by NIDRIVER.

IO$_SETMODE!IO$_STARTUP

Issuing a QIO with a "func" argument of IO$_SETMODE!IO$_STARTUP causes NIDRIVER to begin controller operation. NIDRIVER will allocate necessary VAX/VMS resources and pass a GO ON-LINE command to the NI1010 controller. The controller and driver will now be in a state to process commands and receive packets.

One can modify NIDRIVER's resource allocation strategy at run-time. To change strategy, one's program must supply (in the IO$_SETMODE! IO$_STARTUP QIO) the address of a quadword characteristics buffer in parameter P1 and the size in bytes (always 16) of the characteristics buffer in P2. The first longword (32-bit word) of the characteristics buffer is interpreted by NIDRIVER as follows:

<3:0> the maximum number of receive buffers that NIDRIVER will pass to the controller with SUPPLY RECEIVE BUFFER commands (maximum of 16). NIDRIVER will allocate 5 Unibus Adapter map registers for each as a result of this call. If this field is zero, NIDRIVER will use a default value of 4 and allocate 20 map registers for receive operations. (Five additional map registers are ALWAYS allocated for command DMA.)

<31:4> RESERVED (must be zero)

The second longword of the characteristics buffer is interpreted by NIDRIVER as follows:

<0:0> = 0 (Default) Allocate a Buffered Data Path only when needed to process command operations

that perform DMA. Deallocate the Buffered Data Path immediately after the command is finished.

<0:C> = 1 Permanently allocate a Buffered Data Path tc be used for command operations that perfcrm DMA.

<31:1> RESERVED (must be zero)

IO$_SETMODE!IO$_SHUTDOWN

Issuing IO$_SETMODE!IO$_SHUTDOWN causes NIDRIVER to shut down netwcrk operaticns. NIDRIVER passes a RESET command to the ccntrcller. All outstanding receive (IO$_READLBLK) requests will finish with a status of SS$_ABORT. Any allocated Buffered Data Path will be deallocated. Any supplied characteristics buffer is ignored for this call.

IO$_WRITELBLK

Issuing IO$_WRITELBIK causes the packet defined by QIO parameters P1 and P2 to be transmitted onto the Ethernet. P1 is the address of a WORD ALIGNED preformatted packet in memory. P2 is the length in bytes of the preformatted packet. P2 must be greater than 0 and less than 1536 (decimal) and even. The packet must follow the format described in Figure 4.2. The QIO request will remain cutstanding until the packet is successfully transmitted onto the Ethernet (cr an error occurs). The IO$_WRITEPBLK functicn performs the same operation as the IO$_WRITELBLK function.

IO$_READLBLK

Issuing IC$_READLBLK causes the buffer defined by QIO parameters P1 and P2 (address and size in bytes, respectively) to be used tc hold the next received packet (or packet fragment if buffer-chaining takes place). The buffer pointed to by P1

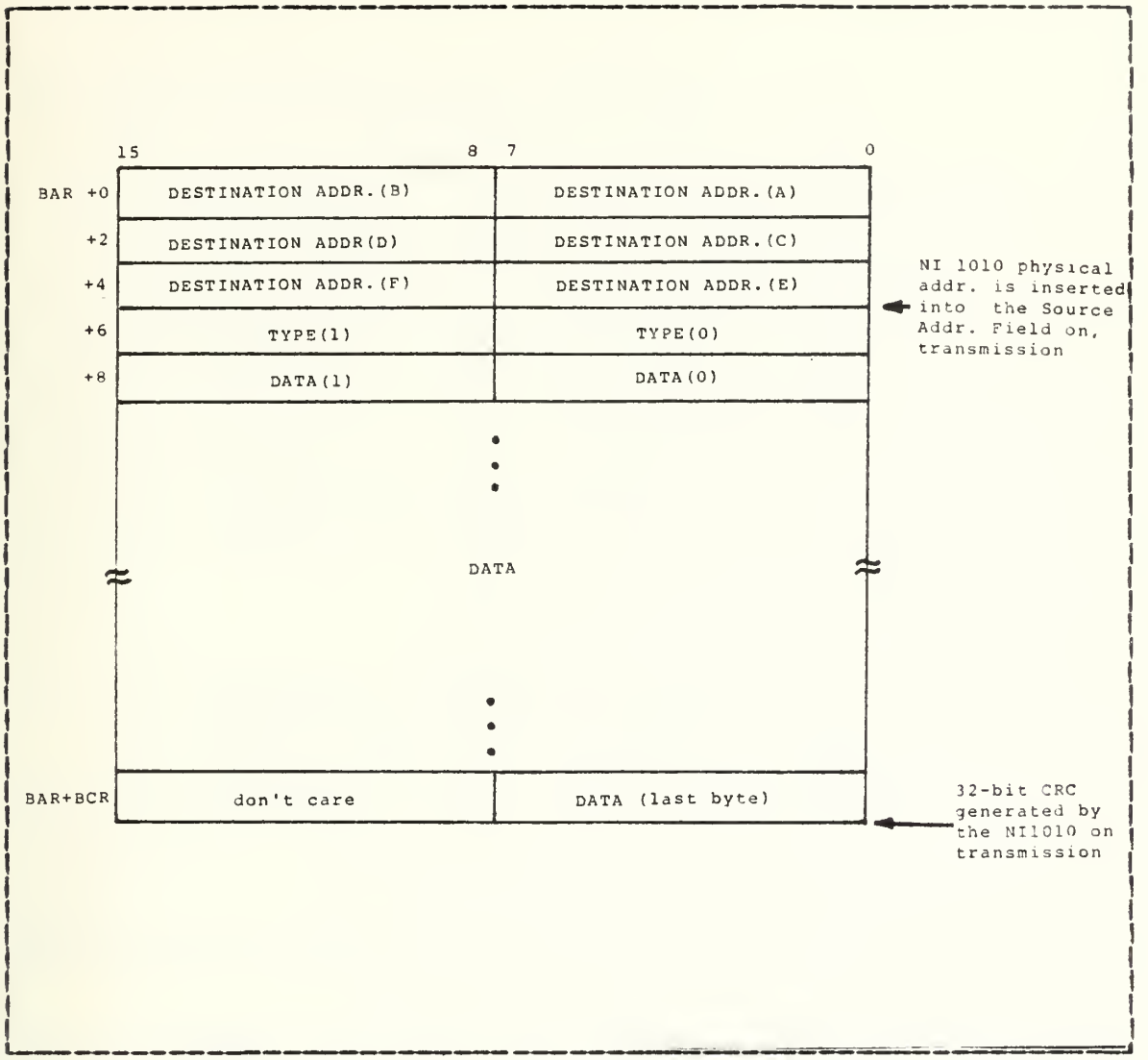


Figure 4.2 Transmit Packet Format.

must be WORD ALIGNED. The buffer size passed in P2 must be greater than 0, less than 1536 (decimal), and even. Packet placement into buffers is strictly FIFO; that is, the oldest outstanding buffer will receive the next incoming packet. The format of the received packet is shown in Figure 4.3. The QIO request will remain outstanding until a packet (or packet fragment) is accessed directly into the associated buffer.

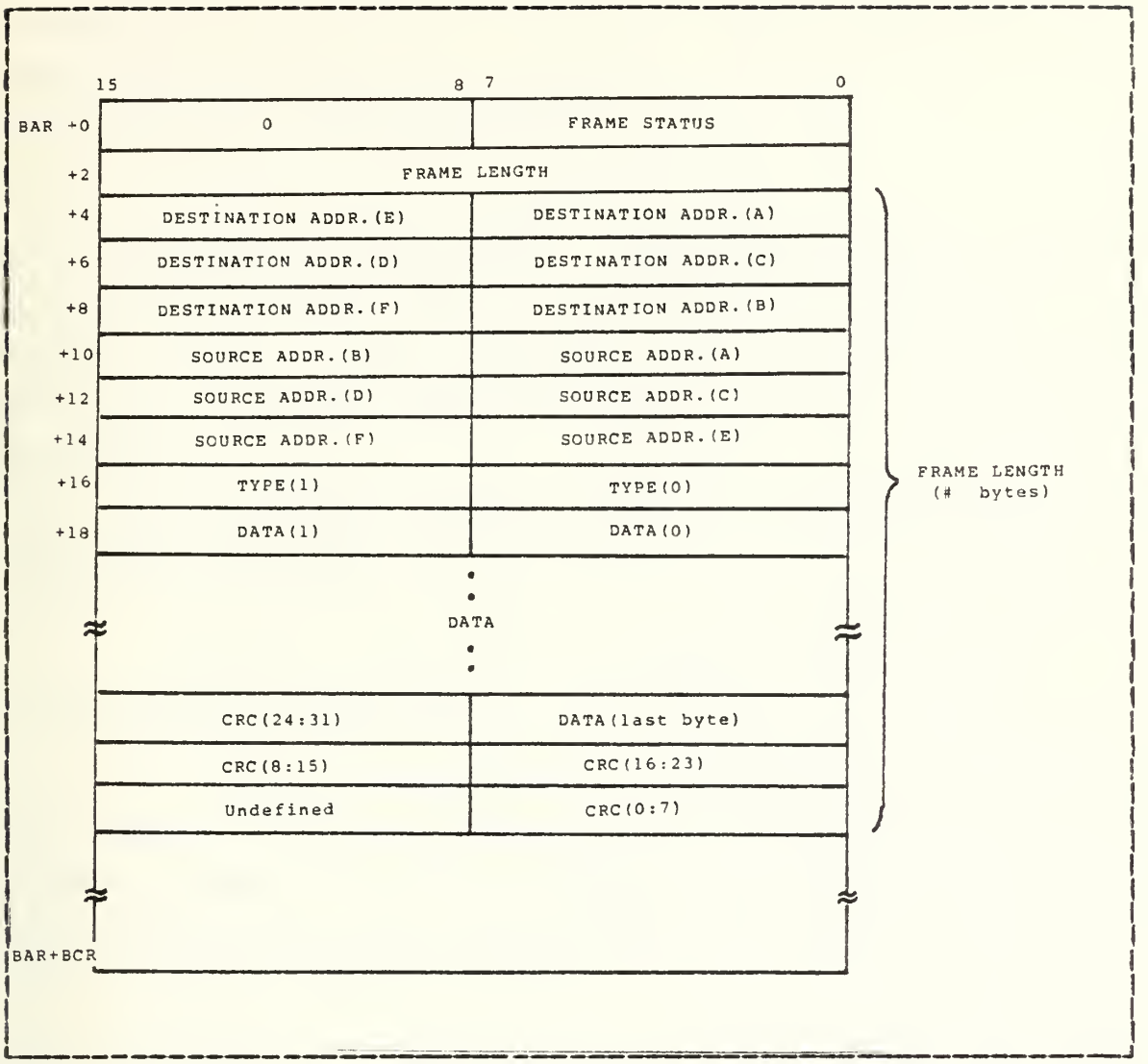


Figure 4.3 Receive Packet Format.

Controller Specific Functions

Because no existing VAX/VMS function codes correspond to NI1010 specific operations such as LOAD MULTICAST or SET PROMISCUOUS MODE, NIDRIVER supports driver-specific function codes. These codes are constructed by passing the controller-specific command in the "function modifier" field of the I/O function value. The function value "code" field will be IO$_READLBLK, IO$_WRITELBLK, or IO$_SEEK, depending

on whether the controller-specific command performs direct memory access or not. For example, the following function value specifies LOAD MULTICAST:

X20 ! < 052 @ 6> = XAA0

As a programming convenience, INTERLAN provides symbolic names which can be used in the function argument of QIO service calls. File NIDEF.MAR can be used with MACRO programs and file NIDEF.FOR can be used with FORTRAN programs.

One can use these definitions in a FORTRAN program by including the line:

INCLUDE '_DRA0:[NPSSYS.INTERLAN]NIDEF.FOR'

in the FORTRAN source code.

(2). I/O Completion. One should always supply the address of a quadword I/O status Block (IOSB) IN THE "iosb" argument of the QIO request. On I/O completion, the IOSB will contain not only VAX/VMS status, but also controller specific status as well.

VAX/VMS status is returned in bits <15:0> of the first IOSB longword. Bits <31:16> of the first IOSB longword do not contain any meaningful imformation. If the returned VAX/VMS status is SS$_NORMAL, Normal Successful Completion, bits <3:0> of the second IOSB longword will contain the Command Status Code from the controller. Refer to the NI1010 Unibus Ethernet Communications Controller User Manual [Ref. 4] for a complete description of the controller status codes. Appendix F describes which Command Status Codes can be expected for each QIO request. Bits <31:4> of the second IOSB longword do not contain meaningful information.

C. DESIGNING PROCEDURE

Since NS2030 Device Driver is intended to be used in VAX/VMS mini-computer, the design is based on Digital Equipment Corporation's Network (DECNET) rather than the ISO Reference Model mentioned in Chapter II. However, the

Layer	ISO	DECNET
7	Application	Application
6	Presentation	
5	Session	(None)
4	Transport	Network Services
3	Network	Transport
2	Data link	Data link Control
1	Physical	Physical

Figure 4.4 Mapping between ISO Model and DECNET.

layering concept is still used in DECNET. The approximate mapping between ISO Reference Model and DECNET is shown in Figure. 4.4.

DECNET has only five layers. The physical layer, data link layer, transport layer, and network services layer correspond almost exactly to the lowest four ISO layers. However, the agreement breaks down at layer 5, since DECNET

has nc session layer, and the remaining layer, the application layer, is a mixture of the ISO presentation and application layers.

Apparently, the NS2030 has covered both network and transport layers, thus, the only layer left to be developed is the application layer.

1. Steps in developing the application layer

With the suggestion from the NS2030 VAX/VMS Device Driver and Diagnostic User Manual [Ref. 5], VAX-FORTRAN programming language is chosen to be used in developing the application layer. The other reason to use FORTRAN is because of the provided function argument of QIO service call in NIDEF.FOR file in the Driver Routine. This makes it easier for the programmer to issue commands to the NI1010 Unibus Ethernet Communications Controller Board. Steps in developing application layers are as follows:

1) All available system service routines in VAX/VMS involving I/O operation are studied. Some of the very important routines which are used in developing the program are included in Appendix A.

2) The first experiment is to check whether the program can really instruct the NI1010 Board what to do. This is done by writing a program that will send out a message to the NI1010 Board and direct the board to send the message back to itself,i.e., send the message from memory to the transmit buffer and send that same message back to the receive buffer of NI1010 Board. In order to do this, the NI1010 Board must be put in the INTERNAL LOOP BACK MODE. The detail of command descriptions available to be used with NI1010 Board can be found in NI1010A Unibus Ethernet Communications Controller User Manual [Ref. 4]. The program that is developed for this experiment is included in Appendix B.

3) The second experiment is to do the same thing except that this time the message will be sent out to the transceiver and onto the coax cable, but the Destination Address field contains the address of the board itself (02-07-01-00-07-7F), thus this message will come back to the receive buffer again. This is called "EXTERNAL LOOP BACK MODE".

4) After the first two experiments are completed successfully, the Destination Address field is changed to that of the NS3010 Board implemented in the MDS system. We have two NS3010 Boards, one with address 02-07-01-00-04-0A and the other with address 02-07-01-00-03-EA.

5) The next step is to transfer a file. The same type of experiment which has been done in sending and receiving the message is used. The DOWNLOAD and UPLOAD procedure in the VAX/VMS are studied. All the FORTRAN statements used in file operation can be found in VAX-11 FORTRAN User's Guide [Ref. 8].

2. Method to Overcome Frame Sequencing

It has been mentioned earlier that the NI1010 Board does not have a capability to preserve the order of messages. Therefore the design of the application layer protocol should take this matter into consideration.

The solution is that the convention of communicating between any two computer systems should be made such that both stations will be able to know each other's status. The convention of communicating has been established as follows:

a). The receiving station will send an acknowledge message every time a frame is received successfully. If an error should occur, no acknowledge message will be sent.

b). The sending station should wait for the acknowledge message from the receiving station for a sufficient amcut of time (protocol in VAX/VMS is set up for 5 seconds), if there is no acknowledge message within this period of time it will retransmit the same frame again and wait for the acknowledge message. The same frame is transmitted for the total of 3 times (1 transmission and 2 retransmissions) before the transmit process will be aborted.

c). The convention used to differentiate whether the frame is carrying a message or a file is established by the use of the available Type field (2 bytes) of each frame. It has been set up as shown in Table I.

TABLE I

Type Field Protocol: (All in Hexadecimal)

BYTE1	BYTE2	FUNCTION
00	00	console message
00	FF	acknowledge message
0F	00	file transfer-first frame
0F	01	file transfer-intermed frame
0F	0F	file transfer-1 record file
0F	FF	file transfer-last frame

D. IMPLEMENTATION

The final software protocol (application layer protocol) whose source code is shown in Appendix C is now available in VAX/VMS for public use. This program is in the file ETHERNET.FOR. A user who wants to transfer files or messages between VAX/VMS and MDS systems can do so by following the instructions given in VAX/VMS-MDS Ethernet Local Communication Network User Manual included in Appendix D.

V. CONCLUSION

The principal goals of this thesis were met. The developed software protocol was tested with the actual transfer of messages and files between VAX-11/780 under VMS operating system and MDS System under CP/M-80 operating system. A file as large as 43 Kbytes was transferred roughly in less than 42 seconds.

At present, the program is available in VAX/VMS public user account under user name "INTERLAN" with password "VMS". The VAX/VMS-MDS Ethernet Local Communication Network User Manual is also available in the file "VMSMDS.DAT". The content in this file is exactly the same as the content in Appendix D in this thesis. Users who want to do the message or file transfer can get the hard copy of this file by simply logging into the VAX/VMS under user name and password mentioned above and printing the file. Then the steps in the manual must be followed.

The files in public user account are:

ETHER1.FOR (source code)
ETHER1.EXE (executable code)

This is a program to transfer a message in the INTERNAL LOOPBACK mode.

SENDMSG.FOR
SENDMSG.EXE

This is a program to send a message, typed in from the terminal, from the VAX/VMS to the MDS System. It retransmits the same message 3 times with the interval of 5 seconds before the transmit process is aborted if there is no acknowledge signal from the receiving station.

GETMSG.FOR
GETMSG.EXE

This program waits for the message intended for the VAX/VMS. It sends back the acknowledge signal to the sending station every time it receives a frame successfully. The received message is displayed on the screen.

DOWNLOAD.FOR
DOWNLOAD.EXE

This program is used to transfer a specified file from VAX/VMS to MDS System. It will wait for an acknowledge signal from the receiving station after every frame has been transmitted. The same frame will be transmitted 3 times with the interval of 5 seconds before the transmit process is aborted, if there is no acknowledge signal from the receiving station. The file is transferred by a record of 128 bytes so it would match the characteristics of CP/M records.

UPLOAD.FOR
UPLOAD.EXE

It is a program used to receive the incoming file from the MDS System. It sends an acknowledge signal to the sending station for every successfully received frame. This program puts VAX/VMS into a ready-to-receive-file mode until control-Y key is pressed.

ETHERNET.FOR
ETHERNET.EXE

This program is a combination of all the programs mentioned above. When executed, VAX/VMS will be ready to receive any message in the network which is intended for the VAX/VMS. The message will be interpreted

whether it is a ordinary message, a request transferring of file, or a request receiving of file.

1).If the message is an ordinary message, the program prints that message on the screen, sends an acknowledge signal to the sending station, and is ready to receive another message.

2).If the message is a request for transferring a file, the program sends back an acknowledge signal and transfers a specified file to the sending station until the whole file has been transferred successfully. The request for transferring a file message should include the filename and filetype, FN.FT, of the file which the requesting station wants to receive. If the public user account does not have the specified file, and error message will be sent to the requesting station to notify the user.

3).If the received message is a request for receiving a file, the program will send an acknowledge message together with instructions to the user of the requesting station to open a new file under the specified FN.FT, receive the incoming file until its all done, then send a message to the sending station that the whole file has been received successfully and then put VAX/VMS back to ready-to-receive-message mode.

All of these files can be copied by any users by typing the following commands:

```
$Copy
$From:      _DRA1:[INTERLAN]FN.FT
```

```
$Tc:        NFN.NFT
```

where FN.FT is the filename.filetype of the file to be copied from, and NFN.NFT is the filename.filetype of the file to be copied to. The NFN.NFT will appear in the user's directory after the above sequence of commands have been executed. It is necessary that the file type of the new file should be the same as the old file.

Future research with VAX/VMS Ethernet Software Protocol should concentrate on trying to make the MDS System terminal act like a virtual terminal of VAX/VMS. There are system service routines available in VAX/VMS which support this capability. Anyone who is interested to do a further research in this field can get all the information about these routines from Mr.Albert Wong, VAX professional staff, in Rm. SP505. The modifications can be made without any changes in the present programs since this program is designed with a layering concepts of the network protocol.

Another direction of research is to expand the network so that VAX/VMS can also communicate with other systems under different operating systems such as ISIS II or MCORTEX.

APPENDIX A
VAX/VMS SYSTEM SERVICE ROUTINES

The followings are the VAX/VMS System Service Routines which are used in developing the application layer protocol for the Ethernet Local Area Network.

$ASSIGN

$ASSIGN - ASSIGN I/O CHANNEL

The Assign I/O Channel system service (1) provides a process with an I/O channel so that input/output operations can be performed on a device, or (2) establishes a logical link with a remote node on a network.

High-level Language Format

SYS$ASSIGN (devnam,chan,[acmode],[mbxnam])

devnam

Address of character string descriptor pointer to the device name string. The string may be either a physical device name or a logical name. If the device name contains a colon, the colon and the characters that follow it are ignored. If the first character in the string is a underscore character(_), the name is considered a physical device name. Otherwise, the name is considered a logical name and logical name translation is performed until either a physical device name is found or the system default number of translations has been performed.

If the device name contains a double colon (::), the system assigns a channel to the first available network

device (NET:) and performs an access function on the network.

chan

Address of a word to receive the assigned channel number.

acmode

Access mode to be associated with the channel. The most privileged access mode used is the access mode of the caller. I/O operations on the channel can only be performed from equal and more privileged access modes.

mbxnam

Address of a character string descriptor pointing to the logical name string for the mailbox to be associated with the device, if any. The mailbox receives status information from the device driver.

An address of 0 implies no mailbox; this is the default value.

Notes

1) For details on how to use $ASSIGN in conjunction with network operations, see the DECnet-VAX User's Guide [Ref. 9].

2) Only the owner of a device can associate a mailbox with the device (the owner is the process that has allocated the device, either implicitly or explicitly), and only one mailbox can be associated with a device at a time. If a mailbox is associated with a device, the device driver can send messages containing status information to the mailbox, as in the following cases:

a).If the device is a terminal, a message indicates dial-up, hang-up, or the reception of unsolicited input.

b).If the target is on a network, the message may indicate that the network is connected or initiated, or whether the line is down.

For details on the message format and the information returned, see the VAX/VMS I/O User's Guide [Ref. 7].

3) Channels remain assigned until they are explicitly deassigned with the Deassign I/O Channel ($DASSGN) system service, or, if they are user-mode channels, until the image that assigned the channel exits.

4) The $ASSIGN service establishes a path to device, but does not check whether the caller can actually perform input/output operations to the device. Privilege and protection restrictions may be applied by the device drivers. For details on how the system controls access to devices, see the VAX/VMS I/O User's Guild [Ref. 7].

$QIOW - QUEUE I/O REQUEST AND WAIT FOR EVENT FLAG

The Queue I/O Request and Wait for Event Flag system service combines the $QIO and $WAITFR (Wait for Single Event Flag) system services. It can be used when program must wait for I/O completion.

High-level Language Format

```
SYS$QIOW ([efn],chan,func,[iosb],[astadr],[astprm],
          [p1],[p2],[p3],[p4],[p5],[p6])
```

efn

Number of the event flag that is to be set at request completion. If not specified, it defaults to 0.

chan

Number of the I/O channel assigned to the device to which the request is directed.

func

Function code and modifier bits that specify the operation to be performed. The code is expressed symbolically.

iosb

Address of quadword I/O status block that is to receive final completion status.

astadr

Address of the entry mask of an AST service routine to be executed when the I/O completes. If specified, the AST routine executes at the access mode from which the $QIO service was requested.

astprm

AST parameter to be passed to the AST completion routine.

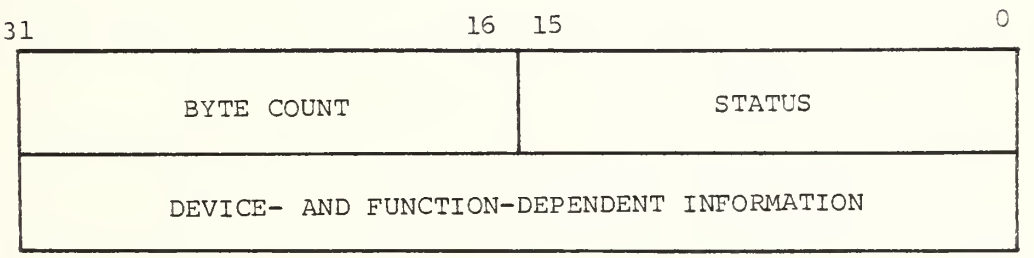
p1 to p6

Optional device- and function-specific I/O request parameters.
The first parameter may be specified as p1 or p1v, depending on whether the function code requires an address or a value, respectively. If the keyword is not used, p1 is the default; that is, the argument is considered an address.
P2 through Pn are always interpreted as values.

Notes

1) The specified event flag is set if the service terminates without queuing an I/O request.

2) The I/O status block has the following format:



a).status - completion status of the I/O request.

b).byte count - Number of byte actually transferred. Note that for some devices this contains only the low-order word of the count.

c).device- and function-dependent information - Varies according to device and operation being performed.

The information returned for each device and function code is documented in the VAX/VMS I/O User's Guide [Ref. 7].

3) Many services return character string data and write the length of the data returned in a word provided by the caller. Function codes for the $QIOW system service (and the LENGTH argument of the $OUTPUT system service) require length specifications in longwords (32-bit word). If lengths returned by other services are to be used as input parameters for $QIOW requests, a longword should be reserved to ensure that no error occurs when $QIOW reads the length.

4) For information on performing input and output operations on a network, see the DECnet-VAX User's Guide [Ref. 9].

$BINTIME

$BINTIME _ CONVERT ASCII STRING TO BINARY TIME

The Convert ASSCII String to Binary Time system service converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the Set Timer ($SETIMR) or Schedule Wakeup ($SCHDWK) system services.

High-level Language Format

SYS$BINTIM (timbuf ,timadr)

timbuf

Address of a character string descriptor pointing to the buffer containing the absolute or delta time to be converted. The required formats of the ASCII strings are described in the Notes, below.

timadr

Address of a quadword that is to receive the converted time in 64-bit format.

Notes

1) The $BINTIM service executes at the access mode of the caller and does not check whether address arguments are accessible before it executes. Therefore, an access violation causes an exception condition if the input buffer or buffer descriptor cannot be read or the output buffer cannot be written.

2) This service does not check the length of the argument list, and therefore cannot return the SS$_INSFARG (insufficient arguments) error status code. If the service does not receive enough arguments (for example, if one omits

required commas in the call), one might not get the desired result.

3) The required ASCII input strings have the format:

Absolute Time: dd-mmm-yyyy hh:mm:ss.cc

Delta Time: ddd hh:mm:ss.cc

Field	Length (bytes)	Contents	Range of values
dd	2	day of month	1 - 31
-	1	hyphen	Required syntax
mmm	3	month	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
-	1	hyphen	Required syntax
yyyy	4	year	1858 - 9999
blank	n	blank	Required syntax
hh	2	hour	00 - 23
:	1	colon	Required syntax
mm	2	minute	00 - 59
:	1	colon	Required syntax
ss	2	second	00 - 59
.	1	period	Required syntax
cc	2	hundredths of second	00 - 99
dddd	4	number of days (in 24-hour units)	000 - 9999

Note that month abbreviations must be upper case. In contrast with previous versions of VAX/VMS, the hundredths of second field now represents a true fraction. For example, the string .1 represents ten hundredths of a second (one tenth of a second); the string .01 represents one hundredth of a second. Note also that a third digit can be added to the hundredths of second field; this thousandths of second digit is used to round the hundredths of second value. Digits beyond the thousandths of second digits are ignored.

4) The following syntax rules apply to specifying the ASCII input string:

a). Any of the date and time fields can be omitted.

For absolute time values, the $BINTIM service supplies the current system date and time for nonspecified fields. Trailing fields can be truncated. If

leading fields are omitted, the punctuation (hyphens, blanks, colons, periods) must be specified. For example, the following string results in an absolute time of 12:00 on the current day.

```
__ 12:00:00.00
```

For delta time values, the $BINTIM service defaults nonspecified hours, minutes, and seconds fields to 0. Trailing fields can be truncated. If leading fields are omitted from the time value, the punctuation (blanks, colons, periods) must be specified. If the number of days in the delta time is 0, a 0 must be specified. For example, the following string results in a delta time of 10 seconds.

```
0 ::10
```

Note the space between the 0 in the day field and the two colons.

b). For both absolute and delta time values, there can be any number of leading blanks, and any number of blanks between fields normally delimited by blanks. However, there can be no embedded blanks within either the date or time fields.

The following examples illustrate legal input strings to the $BINTIM system service, and the time represented by the output from the $BINTIM system service (translated through the Convert Binary Time to ASCII String ($ASCTIM) system service). Assume that the current date is 14-JUN-1983 04:15:28.00.

Input to $BINTIM	$ASCTIM output string
-- :50	14-JUN-1983 04:50:28.00
--1984 0:0:0.0	14-JUN-1984 00:00:00.00
9-NOV-1982 12:32:1.1161	9-NOV-1982 12:32:01.12

```
22-APR-1983 16:35:0.0
0 ::.1
0 ::.06
5 3:18:32.068
20 12:
0 5
```

```
22-APR-1983 16:35:00.00
0 00:00:00.10
0 00:00:00.06
5 03:18:32.07
20 12:00:00.00
0 05:00:00.00
```

$SETIMR

$SETIMR - SET TIMER

The Set Timer system service allows a process to schedule the setting of an event flag and/or the queuing of an AST at some future time. The time of the event can be specified as a absolute time or as a delta time.

When the service is invoked, the event flag is cleared (event flag 0, if none is specified).

High-level Language Format

SYS$SETIMR([efn],daytim,[astadr],[reqidt])

efn

Event flag number of the event flag to set when the time interval expires. If not specified, it defaults to 0.

daytim

Address of the quadword expiration time. A positive time value indicates an absolute time at which the timer is to expire. A negative time value indicates an offset (delta time) from the current time.

astadr

Address of the entry mask of a AST service routine to be called when the time interval expires. If not specified, it defaults to 0, indicating no AST is to be queued.

reqidt

Number indicating a request identification. If not specified, it defaults to 0. A unique request

identification can be specified in each set timer requests, or the same identification can be given to related timer requests. The identification can be used later to cancel the timer request(s). If an AST service routine is specified, the identification is passed as the AST parameter.

Notes

1) The access mode of the caller is the access mode of the request and of the AST.

2) If a specified absolute time value has already passed, the timer expires at the next clock cycle (that is, within 10 milliseconds).

3) The Convert ASCII String to Binary Time ($BINTIM) system service converts a specified ASCII string to the quadword time format required as input to the $SETIMR service.

$WAITFR

$WAITFR - WAIT FOR SINGLE EVENT FLAG

The Wait for Single Event Flag system service tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.

High-level Language Format

```
SYS$WAITFR(efn)
```

efn

Number of the event flag for which to wait.

Notes

The wait state caused by this service can be interrupted by an asynchronous system trap (AST) if (1) the access mode at which the AST executes is more privileged than or equal in privilege to the access mode from which the wait was issued and (2) the process is enabled for ASTs at that access mode.

When the AST service routine completes execution, the system repeats the $WAITFR request. If the event flag has been set, the process resumes execution.

$CANTIM

$CANTIM - CANCEL TIMER

The Cancel Timer Request system service cancels all or a selected subset of the Set Timer requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the Set Timer ($SETIMR) system service. If more than one timer request was given to the same request identification, they are all canceled.

High-level Language Format

SYS$CANTIM ([reqidt],[acmode])

reqidt

Request identification of the timer request(s) to be canceled. A value of 0 (the default) indicates that all timer requests are to be canceled.

acmode

Access mode of the request(s) to be canceled. The most privileged access mode used is the access mode of the caller. Only those timer requests issued from an access mode equal to or less privileged than the resultant access mode are canceled.

Notes

Outstanding timer requests are automatically canceled at image exit.

APPENDIX B
SOURCE CODE FOR EXPERIMENTS

All the source code in this appendix was developed from the step-by-step design of the Ethernet Software Protocol. Each of the programs includes a brief explanation of the function when it is executed.

```
      PROGRAM ETHER1
C
C     TRANSFER MESSAGE/INTERNAL LOOPBACK.

      character*26      text /'This message will be sent.'/
      integer*2         iosb(2)
      integer*4         nichan, sys$qiow, sys$assign
      include           '[dra0:[nossys.interlan]nidef.for'
      include           '($iodef)'
      byte              Tpacket(136), Rpacket(146)

C     Assign destination address:
      Tpacket(1)='02'x
      Tpacket(2)='07'x
      Tpacket(3)='01'x
      Tpacket(4)='00'x
      Tpacket(5)='07'x
      Tpacket(6)='7F'x

C     Type assignment: 0000=nsp, etc.
      Tpacket(7)='00'x
      Tpacket(8)='00'x

C     Put data into transmit packet:
      j=9
      do i=1,26
              Tpacket(j)=ichar(text(i:i))
              j=j+1
      end do
      do I=35,136
              Tpacket(I)='00'x
      end do

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))
      type *,' A istat=',istat

C     Start up and go on line:
      istat= sys$qiow(,%val(nichan),
     1                 %val(io$_setmode .or. io$m_startup),
     2                iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))
      type *,' S istat=',istat,' S iosb(1)=',iosb(1)

C     Internal loopback:
      istat=sys$qiow(,%val(nichan),
     1                %val(io$_silm),
     2                iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))
      type *,' I istat=',istat,' I iosb(1)=',iosb(1)

C     Promiscuous:
      istat=sys$qiow(,%val(nichan),
     1                %val(io$_sprm),
     2                iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))
```

```
C     Receive on error:
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_sroem),
     2               iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))

C     Transmit packet:
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_writelblk),
     2               iosb,,,
     3               %ref(Tpacket),%val(136),,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))
      type *,' T istat=',istat,' T iosb(1)=',iosb(1)
      type *, Tpacket

C     Load transmit data and send:
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_ltd),
     2               iosb,,,
     3               %ref(Tpacket),%val(136),,,,)
      if(.not. istat) call lib$stop(%val(istat))
      type *,' Message is being transmitted....'

C     Receive same packet:
      type *,' start receiving'
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_readlblk),
     2               iosb,,,
     3               %ref(Rpacket),%val(140),,,,)
      type *, ' iosb(2)=',iosb(2)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))
      type *,' R istat=',istat,' R iosb(1)=',iosb(1)
      type *,Rpacket
      i=19
      do while (Rpacket(i).ne.ichar('.'))
              type *,char(Rpacket(i))
              i = i+1
      end do
      call exit
      end
```

```
      PROGRAM SENDMSG
C
C     ACTUAL SENDING OF MESSAGE.
C     WAIT FOR ACKNOWLEDGE.
C     IF NOT ACKNOWLEDGE IN 5 SEC, RETRANSMIT.
C     IF NOT ACKNOWLEDGE IN 10 SEC, ABORT TRANSMISSION.

      external          abort
      integer*2         iosb(2),addr
      integer*4         nichan, sys$qiow, sys$assign
      integer*4         sys$bintim, sys$setimr, sys$waitfr
      integer*4         systime(2),time(2)
      include           'edra0:[nossys.interlanlnidef.for'
      include           '($iodef)'
      byte              pad(100)
      byte              Tpacket(136),text(128),Rpacket(150)

C     Assign a channel to NIA0:
10    istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Start up and go on line:
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_setmode .or. iof_startup),
     2               iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).ne.1) call lib$stop(%val(iosb(1)))

C     Assign destination address:
      Tpacket(1)='02'x
      Tpacket(2)='07'x
      Tpacket(3)='01'x
      Tpacket(4)='00'x

C     Interact with user:
20    type *, ' Select Net Address of Destination:'
      type *, ' MDS System 00-04-0A : type"1"'
      type *, ' MDS System 00-03-EA : type"2"'
      read(5,11)addr
11    format(a1)
      if (addr.eq.'1') then
              Tpacket(5)='04'x
              Tpacket(6)='0A'x
      else if (addr.eq.'2') then
              Tpacket(5)='03'x
              Tpacket(6)='EA'x
      else
              goto 20
      end if
      type *,'Input message(128 char max)end with ` '
      read(5,22)text
22    format(128a1)

C     Assign type field:
      Tpacket(7)='00'x      ! indicate that it is a message
      Tpacket(8)='00'x      ! don't care

C     Put data into transmit packet:
      j=9
      do i=1,28
              Tpacket(j)=text(i)
```

```
                j=j+1
        end do

C       Transmit packet:
30      istat=sys$qiow(,%val(nichan),
        1               %val(io$_writelblk),
        2               iosb,,,
        3               %ref(Tpacket),%val(135),,,,)
        if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
        if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
        type *,' Message is being transmitted.....'

C       Load transmit data and send:
        istat=sys$qiow(,%val(nichan),
        1               %val(io$_eltas),
        2               iosb,,,,,,,,)
        if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
        if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C       Wait for 10 seconds and abort:
        call sys$bintim('0 ::10.0',time)
        call sys$setimr(,time,abort,)

C       Wait for 5 second and retransmit:
        call sys$bintim('0 ::5.0',systime) ! retransmit
        call sys$setimr(1,systime,,)       ! after 5 sec.

C       Receive acknowledge:
        istat=sys$qiow(,%val(nichan),
        1               %val(io$_readlblk),
        2               iosb,,,
        3               %ref(Rpacket),%val(150),,,,)
        if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
        if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
C       Check the second type field if = FF hex:
        if (Rpacket(18).eq.'FF'x) then
                i = 19
                do while (Rpacket(i).ne.ichar('`'))
                        i = i+1
                end do
                write(6,33)(Rpacket(j),j=19,i-1)
33              format(' ',<i>a1)
                call sys$cantim( , )  ! cancel timer
                Rpacket(18)='00'x
                goto 20
        end if
        call sys$waitfr(1)
        goto 30
        end

CCCC
```

```
      PROGRAM GETMSG
C
C     ACTUAL RECEIVING OF MESSAGE.
C     SEND BACK ACKNOWLEDGE MSG.(2nd type field=FF hex).
C     IF RECEIVED BAD FRAME, DO NOTHING BUT WAIT TO RECEIVE.

      character*23     msg1/' Receive successfully.'/
      integer*2        iosb(2)
      integer*4        nichan, sys$qiow, sys$assign
      include          'edra0:[nossys.interlan]nidef.for'
      include          '($iodef)'
      byte             pad(100)
      byte             Rpacket(150),Tpacket(130)
      byte             dfilnam(40),sfilnam(40)

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) type *, ' Assign error!'

C     Start up and go on line:
      istat= sys$qiow(,%val(nichan),
     1          %val(io$_setmode .or. io$m_startup),
     2          iosb,,,,,,,)
      if(.not.istat) type *,' Istat start up error!'
      if(iosb(1).lt.0) type *, ' Start up error!'

C     Receive incoming message:
10    type *,' Ready to receive.......'
      istat=sys$qiow(,%val(nichan),
     1                %val(io$_readlblk),
     2                iosb,,,
     3                %ref(Rpacket),%val(150),,,,)
      if(.not.istat) then
              type *,' Send msg istat receive error'
              goto 10
      end if
      if(iosb(1).lt.0) then
              type *,' Send msg VAX/VMS receive error'
              goto 10
      end if
      if(iosb(2).eq.1) then
              type *,' Send msg receive block CRC error'
              goto 10
      end if
      if(iosb(2).eq.2) then
              type *,' Receive block alignment error'
              goto 10
      end if
      if(iosb(2).eq.4) then
              type *,' Send msg missing receive block'
              goto 10
      end if
      if(iosb(2).eq.17) then
              type *,' Send msg DMA received block fail'
              goto 10
      end if
      i=19
      do while (Rpacket(i).ne.ichar('*'))
              i = i+1
      end do
      write(6,11)(Rpacket(j),j=19,i-1)
```

```
11      format(' ',<i>a1)
        type *,' Received successfully.'

C     Assign destination address:
        Tpacket(1)='02'x
        Tpacket(2)='07'x
        Tpacket(3)='01'x
        Tpacket(4)=Rpacket(14)
        Tpacket(5)=Rpacket(15)
        Tpacket(6)=Rpacket(16)

C     Assign type field:
        Tpacket(7)='00'x    ! indicate that it is a message
        Tpacket(8)='FF'x    ! acknowledge signal

C     Put data into transmit packet:
        j=9
        do i=1,23
                Tpacket(j)=ichar(msg1(i:i))
                j=j+1
        end do

C     Transmit packet:
        istat=sys$qiow(,%val(nichan),
        1                %val(io$_writelblk),
        2                iosb,,,
        3                %ref(Tpacket),%val(136),,,,)
        if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
        type *,' Acknowledge is being transmitted......'

C     Load transmit data and send:
        istat=sys$qiow(,%val(nichan),
        1                %val(io$_ltds),
        2                iosb,,,,,,,,)
        if(iosb(1).lt.0) type *,' Ether xmit error!'
        if(iosb(2).ne.0) type *,' Controller xmit error!'
        goto 10
        end
```

```
      PROGRAM UPLOAD
C
C     ACTUAL RECEIVING OF FILE.
C     DISCARD LAST FRAME.
C     CHECK FOR 1 RECORD FILE.
C     SEND BACK ACKNOWLEDGE MSG.(2nd type field=FF hex).
C     IF RECEIVE BAD FRAME, DO NOTHING BUT WAIT TO RECEIVE
C     SAME FRAME AGAIN.

      character*24    msg1/' Received successfully.'/
      integer*2       iosb(2),done
      integer*4       nichan, sys$qiow, sys$assign
      include         'edra0:[npssys.interlan]nidef.for'
      include         '($iodef)'
      byte            pad(100)
      byte            Rpacket(150),dfilnam(40)
      byte            Tpacket(136)

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Start up and go on line:
      istat= sys$qiow(,%val(nichan),
     1         %val(io$_setmode .or. io$m_startup),
     2         iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) type *,'Controller Start up error'

C     Initialize flag:
      done=0

C     Interact with user:
2     type *,' Destination filename?'
      read(5,52)ln,dfilnam
      dfilnam(ln+1)=0
52    format(q,40a1)
      open(name=dfilnam,unit=2,organization='sequential',
     1 type='new',carriagecontrol='list',iostat=ios,err=7)
      goto 10
C7    type *,' Open file fail! Try again'
7     call lib$stop(%val(ios))
      goto 2

C     Receive file:
10    type *,' Ready to receive....'
20    istat=sys$qiow(,%val(nichan),
     1               %val(io$_readlblk),
     2               iosb,,,
     3               %ref(Rpacket),%val(150),,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) type *,' Controller Receive error?'
      if (Rpacket(18).eq.'0F'x) then   ! a 1 record file
            write(2,550)(Rpacket(j),j=19,146)
            done=1
            goto 40
      end if
      if (Rpacket(18).eq.'FF'x) then
            done=1
```

```
                goto 40
        end if
        write(2,550)(Rpacket(j),j=19,146)
550     format(128a1)

C       Assign destination address:
40      Tpacket(1)='02'x
        Tpacket(2)='07'x
        Tpacket(3)='01'x
        Tpacket(4)=Rpacket(14)
        Tpacket(5)=Rpacket(15)
        Tpacket(6)=Rpacket(16)

C       Assign type field:
        Tpacket(7)='00'x ! indicate that it is a message
        Tpacket(8)='FF'x ! acknowledge signal

C       Put data into transmit packet:
        j=9
        do i=1,24
                Tpacket(j)=ichar(msg1(i:i))
                j=j+1
        end do

C       Transmit packet:
        istat=sys$qiow(,%val(nichan),
        1                %val(io$_writelblk),
        2                iosb,,,
        3                %ref(Tpacket),%val(130),,,,)
        if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
        type *,' Acknowledge is being transmitted.....'

C       Load transmit data and send:
        istat=sys$qiow(,%val(nichan),
        1                %val(io$_eltds),
        2                iosb,,,,,,,,)
        if(iosb(1).lt.0) type *,' Ether xmit error!'
        if(iosb(2).ne.0) type *,' Controller xmit error!'
        if (done.eq.1) goto 30
        goto 20

30      close(unit=2)
        type *,' Receive completed!'
        call exit
        end
```

```
      PROGRAM DOWNLOAD
C
C     ACTUAL TRANSFER FILE/INTERACTIVE.
C     RETRANSMIT SAME FRAME IF NO ACKNOWLEDGE IN 5 SEC.
C     ABORT TRANSMISSION IF NOT RECEIVE ACKNOWLEDGE IN 10 SEC.

      external          abort
      integer*2         iosb(2), addr, count, done
      integer*4         nichan, sys$qiow, sys$assign
      integer*4         time(2),systime(2),sec(2)
      include           'edra0:[nossys.interlan]nidef.for'
      include           '($iodef)'
      byte              pad(100)
      byte              Tpacket(136),sfilnam(40)
      byte              Rpacket(150)

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Start up and go on line:
      istat= sys$qiow(,%val(nichan),
     1          %val(io$_setmode .or. io$_startup),
     2          iosb,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) type *,'Controller Start up error'

C     Assign destination address:
      Tpacket(1)='02'x
      Tpacket(2)='07'x
      Tpacket(3)='01'x
      Tpacket(4)='00'x

C     Interact with user:
10    type *,' Select Net Address of Destination:'
      type *,' MDS System 00-04-0A : type"1"'
      type *,' MDS System 00-03-EA : type"2"'
      read(5,11)addr
11    format(a1)
      if (addr.eq.'1') then
              Tpacket(5)='04'x
              Tpacket(6)='0A'x
      else if (addr.eq.'2') then
              Tpacket(5)='03'x
              Tpacket(6)='EA'x
      else
              goto 10
      end if

C     Assign type field:
      Tpacket(7)='0F'x
      Tpacket(8)='00'x    ! first frame recin(8)=00 nex

C     Initialize flag:
      done = 0

C     Interact with user:
20    type *,' Source filename?'
      read(5,22)ln,sfilnam
      sfilnam(ln+1)=0
```

```
22      format(q,40a1)
        open(name=sfilnam ,unit=1,organization='sequential'
        1       ,type='old',carriagecontrol='list',err=9)
        goto 30
9       type *, ' No such file!  Try again'
        goto 20

C     Transmit packet:
30      count=0
        read(1,33,iostat=ios,end=100,err=101)
        1    (Tpacket(j),j=9,136)
33      format( 128a1)
40      istat=sys$qiow(,%val(nichan),
        1                 %val(io$_writelblk),
        2                 iosb,,,
        3                 %ref(Tpacket),%val(136),,,,)
        if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
        if(iosb(2).ne.0) type *,'Controller Transmit error'
        type *,' File is being transmitted.....'

C     Load transmit data and send file:
        istat=sys$qiow(,%val(nichan),
        1                 %val(io$_ltds),
        2                 iosb,,,,,,,,)
        if(iosb(1).lt.0) type *,' System xmit error!'
        if(iosb(2).ne.0) type *,' Controller xmit error!'

C     Wait for 20 seconds and abort:
        call sys$bintim('0 ::20.0',sec)
        call sys$setimr(,sec,abort,)

C     Wait for 10 seconds and abort:
        call sys$bintim('0 ::10.0',time)
        call sys$setimr(2,time,,)

C     Wait for 5 second and retransmit:
        call sys$bintim('0 ::5.0',systime) ! wait for
        call sys$setimr(1,systime,,)       ! acknowledge 5 sec.

C     Receive acknowledge:
        istat=sys$qiow(,%val(nichan),
        1                 %val(io$_readlblk),
        2                 iosb,,,
        3                 %ref(Rpacket),%val(150),,,,)

C     Check the second type field if = FF hex:
        if (Rpacket(18).eq.'FF'x) then
                call sys$cantim(,)          ! cancel timers.
                Rpacket(18)='00'x           ! clear flag
                if(done.eq.1) goto 50
                Tpacket(8)='01'x            ! middle frame
                goto 30
        end if
        if (count.ne.2) then
                call sys$waitfr(1)
                count=count+1
                goto 40
        end if
        call sys$waitfr(2)
        call exit
101     call lib$stop(%val(ios))
```

```
C     Assign value to end message:
100       do i=9,136
                  Packet(i)='32'x        ! blank char
          end do
          Packet(8)='FF'x                ! lastframe
          done = 1
          goto 40

50        type *,' Transmit completed'
          close(unit=1)
          goto 10
          end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

          SUBROUTINE ABORT

          write(6,222)
222       format(' Abort Transmission')
          call exit
          end
```

APPENDIX C
ETHERNET SOFTWARE PROTOCOL SOURCE CODE

This is the actual source code which is developed to meet the principal goal of this thesis. This program is also available in VAX/VMS public user account under file name "ETHERNET.FOR".

```
      PROGRAM ETHERNET
C
C     ACTUAL RECEIVING OF MESSAGE.
C     SEND BACK ACKNOWLEDGE MSG.(2nd type field=FF hex).
C     IF RECEIVED BAD FRAME, DO NOTHING BUT WAIT TO RECEIVE.
C     CHECK WHAT IS THE REQUEST FROM MDS SYSTEM.
C     IF THE REQUEST IS '@', CALL UPLOAD.
C     IF THE REQUEST IS '!', CALL DOWNLOAD.

      character*28    msg1/' Receive successfully.     '/
      character*28    msg6/' Unrecognized file type.   '/
      integer*2       iosb(2),filtype,exec
      integer*4       systime(2), time(2)
      integer*4       nichan, sys$qiow, sys$assign
      include         '[dra0:[nossys.interlan]nidef.for'
      include         '($iodef)'
      byte            pad(100)
      byte            Rpacket(150),Tpacket(136)
      byte            dfilnam(40),sfilnam(40),ft(3)

C     Assign a channel to NIA0:
10    istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Start up and go on line:
      istat= sys$qiow(,%val(nichan),
      1         %val(io$_setmode .or. io$m_startup),
      2         iosb,,,,,,,,)
      if(.not.istat) call lib$stop(%val(istat))
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))

C     Receive incoming message:
150   type *,' Ready to receive message.....'
      istat=sys$qiow(,%val(nichan),
      1               %val(io$_readlblk),
      2               iosb,,,
      3               %ref(Rpacket),%val(150),,,,)
      if(.not.istat) then
              type *,' Istat receive error'
              goto 10
      end if
      if(iosb(1).lt.0) then
              type *,' VAX/VMS receive error'
              goto 10
      end if
      if(iosb(2).eq.1) then
              type *,' Receive block CRC error'
              goto 10
      end if
      if(iosb(2).eq.2) then
              type *,' Receive block alignment error'
              goto 10
      end if
      if(iosb(2).eq.4) then
              type *,' Missing receive block'
              goto 10
      end if
      if(iosb(2).eq.17) then
              type *,' DMA received block fail'
              goto 10
      end if
```

```
      type *,' Received successfully.'

C     Assign destination address:
      Tpacket(1)='02'x
      Tpacket(2)='07'x
      Tpacket(3)='01'x
      Tpacket(4)='00'x
      Tpacket(5)=Rpacket(15)
      Tpacket(6)=Rpacket(16)

C     Send acknowledge message:

C     Assign type field:
      Tpacket(7)='00'x    ! indicate that it is a message
      Tpacket(8)='FF'x    ! acknowledge signal

C     Put data into transmit packet:
      j=9
      do i=1,30
               Tpacket(j)=ichar(msg1(i:i))
               j=j+1
      end do

C     Transmit packet:
      istat=sys$qiow(,%val(nichan),
      1                %val(io$_writelblk),
      2                iosb,,,
      3                %ref(Tpacket),%val(136),,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      type *,' Acknowledge is being transmitted......'

C     Load transmit data and send:
      istat=sys$qiow(,%val(nichan),
      1                %val(io$_ltds),
      2                iosb,,,,,,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C     Print out incoming message:
      i=19
      if((Rpacket(i).eq.ichar('!')).or.
      1  (Rpacket(i).eq.ichar('i')))then
               do while (Rpacket(i).ne.ichar('/'))
                        i = i+1
                        sfilnam(i-19)=Rpacket(i)
                        dfilnam(i-19)=Rpacket(i)
               end do
               if(((Rpacket(i-3).eq.ichar('e'))
      1        .or.(Rpacket(i-3).eq.ichar('E')))
      2        .and.((Rpacket(i-2).eq.ichar('o'))
      3        .or.  (Rpacket(i-2).eq.ichar('O')))
      4        .and.((Rpacket(i-1).eq.ichar('d'))
      5        .or.  (Rpacket(i-1).eq.ichar('D')))) then
                        exec = 2          ! EOD exec file
               else
                        exec = 1          ! other exec file
               end if
               m = i+1
               do while (Rpacket(m).ne.ichar('`'))
                        ft(m-i)=Rpacket(m)
                        m = m+1
```

```
                end do
                if ((ft(1).eq.ichar('t')).or.
               1    (ft(1).eq.ichar('T'))) then
                  filtype = 1
                else if ((ft(1).eq.ichar('e')).or.
               1         (ft(1).eq.ichar('E'))) then
                  filtype = 2
                else
                  call sendmsg(Rpacket(15),Rpacket(16),msgb)
                  type *,' Unrecognized file type.'
                  goto 150
                end if
        else
                do while(Rpacket(i).ne.ichar('`'))
                   i = i+1
                end do
        end if
        write(6,11)(Rpacket(j),j=19,i-1)
11      format(' ',<i>a1)

C     Check the request message:
        if (Rpacket(19).eq.ichar('!')) then
           sfilnam(i-19)=0
           call download(Rpacket(15),Rpacket(16),sfilnam,
        1                filtype,exec)
        else if (Rpacket(19).eq.ichar('?')) then
           dfilnam(i-19)=0
           call upload(Rpacket(15),Rpacket(16),dfilnam,
        1              filtype,exec)
        end if
        goto 10
        end
```

```
      SUBROUTINE DOWNLOAD(r15,r16,sfilnam,filtype,exec)
C
C     ACTUAL TRANSFER FILE/INTERACTIVE.
C     RETRANSMIT SAME FRAME IF NOT RECEIVE ACKNOWLEDGE IN 5 SEC.
C     ABORT TRANSMISSION IF NOT RECEIVE ACKNOWLEDGE IN 10 SEC.

      character*28    msg2/' No such file! Try again.  '/
      character*28    msg3/' Abort Transmission.       '/
      integer*2       iosb(2),addr,count,done
      integer*2       filtype,exec,max
      integer*4       nichan, sys$qiow, sys$assign
      integer*4       sys$bintim, sys$setimr, sys$waitfr
      integer*4       time(2),systime(2)
      include         'edra0:[nossys.interlan]nidef.for'
      include         '($iodef)'
      byte            oad(100),sendrec(512),sendrec1(1024)
      byte            Tpacket(136),sfilnam(40)
      byte            Rpacket(150),r15,r16

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Print out:
      type *,' Request transferring of file'

C     Print filename:
      write(6,22)(sfilnam(k),k=1,15)
22    format(' ',15a1)

      type *,' to MDS system at address:02 07 01 00',
     1       r15,r16

C     Assign destination address:
      Tpacket(1)='02'x
      Tpacket(2)='07'x
      Tpacket(3)='01'x
      Tpacket(4)='00'x

C     Get the address of requesting station:
      Tpacket(5)=r15
      Tpacket(6)=r16

C     Assign type field:
      Tpacket(7)='0F'x
      Tpacket(8)='00'x          ! first frame

C     Initialize flag:
      done = 0

C     Open file:
      open(name=sfilnam ,unit=1,organization='sequential'
     1     ,type='old',carriagecontrol='list',err=9)

C     File type check point:
      if (filtype.eq.1) then
            goto 30
      else
            goto 200
      end if
```

```
C     Open file error handling:
9        call sendmsg(r15,r16,msg2) ! no such file try again.
         type *,' no such file! Try again.'
         return

C     Send text file:
30       count=0
         read(1,33,iostat=ios,end=100,err=101)
         1    (Tpacket(j),j=9,136)
33       format( 128a1)
         Tpacket(135) = '0A'x                ! <CR>
         Tpacket(136) = '0D'x                ! <LF>
40       istat=sys$qiow(,%val(nichan),
         1                %val(io$_writelblk),
         2                iosb,,,
         3                %ref(Tpacket),%val(136),,,,)
         if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
         if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
         type *,' File is being transmitted.....'

C     Load transmit data and send file:
         istat=sys$qiow(,%val(nichan),
         1                %val(io$_eltds),
         2                iosb,,,,,,,,)
         if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
         if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C     Wait for 15 seconds and abort:
         istat=sys$bintim('0 ::15.0',time)
         if(.not.istat) call lib$stop(%val(istat))
         istat=sys$setimr(%val(2),%ref(time),,)
         if(.not.istat) call lib$stop(%val(istat))

C     Wait for 5 second and retransmit:
         istat=sys$bintim('0 ::5.0',systime)
         if(.not.istat) call lib$stop(%val(istat))
         istat=sys$setimr(%val(1),%ref(systime),,) ! wait for
         if(.not.istat) call lib$stop(%val(istat)) ! 5 sec.

C     Receive acknowledge:
         istat=sys$qiow(,%val(nichan),
         1                %val(io$_readlblk),
         2                iosb,,,
         3                %ref(Rpacket),%val(150),,,,)
         if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
         if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C     Check the second type field if = FF hex:
         if (Rpacket(18).eq.'FF'x) then
                call sys$cantim(,)     ! cancel timers.
                Rpacket(18)='00'x      ! clear flag
                if(done.eq.1) goto 50
                Tpacket(8)='01'x       ! middle frame
                goto 30
         end if
         if (count.ne.2) then
                call sys$waitfr(%val(1))
                count=count+1
                goto 40
         end if
         call sys$waitfr(%val(2))
```

```
      call sendmsg(r15,r16,msg3)    ! abort transmission
      type *,' Abort transmission.'
      return
101   call lib$stop(%val(ios))

C     Assign value to message of the last frame:
100   do i=9,136
            Tpacket(i)='32'x        ! blank char
      end do
      Tpacket(8)='FF'x              ! lastframe
      done = 1
      goto 40

C     Send executable file:
200   if (exec.eq.1) then
         read(1,222,iostat=ios,end=500,err=202)
         1    (sendrec(j),j=1,512)
222      format(512a1)
         max = 5
      else if(exec.eq.2) then
         read(1,333,iostat=ios,end=500,err=202)
         1    (sendrec1(j),j=1,1024)
333      format(1024a1)
         max = 9
      end if
      k = 1
300   count = 0
      do l = 1,128
         if (exec.eq.1) then
            Tpacket(l+8) = sendrec(k*128-(128-l))
         else if (exec.eq.2) then
            Tpacket(l+8) = sendrec1(k*128-(128-l))
         end if
      end do
400   istat=sys$qiow(,%val(nichan),
      1              %val(io$_writelblk),
      2              iosb,,,
      3              %ref(Tpacket),%val(136),,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
      type *,' File is being transmitted.....'

C     Load transmit data and send file:
      istat=sys$qiow(,%val(nichan),
      1              %val(io$_ltds),
      2              iosb,,,,,,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C     Wait for 15 seconds and abort:
      istat=sys$bintim('0 ::15.0',time)
      if(.not.istat) call lib$stop(%val(istat))
      istat=sys$setimr(%val(2),%ref(time),,)
      if(.not.istat) call lib$stop(%val(istat))

C     Wait for 5 second and retransmit:
      istat=sys$bintim('0 ::5.0',systime)
      if(.not.istat) call lib$stop(%val(istat))
      istat=sys$setimr(%val(1),%ref(systime),,) ! wait for
      if(.not.istat) call lib$stop(%val(istat)) ! 5 sec.
```

```
C     Receive acknowledge:
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_readlblk),
     2               iosb,,,
     3               %ref(Rpacket),%val(150),,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C     Check the second type field if = FF hex:
      if (Rpacket(18).eq.'FF'x) then
              call sys$cantim(,)      ! cancel timers.
              Rpacket(18)='00'x       ! clear flag
              if(done.eq.1) goto 50
              Tpacket(8)='01'x        ! middle frame
              k = k+1
              if (k.eq.max) then
                      goto 200
              else
                      goto 300
              end if
      end if
      if (count.ne.2) then
              call sys$waitfr(%val(1))
              count=count+1
              goto 400
      end if
      call sys$waitfr(%val(2))
      call sendmsg(r15,r16,msg3)      ! abort transmission
      type *,' Abort transmission.'
      return
202   call lib$stop(%val(ios))

C     Assign value to message of the last frame:
500   do i=9,136
              Tpacket(i)='32'x        ! blank char
      end do
      Tpacket(8)='FF'x                ! last frame
      done = 1
      goto 400

50    type *,' Transmit completed'
      close(unit=1)
      return
      end
```

```
      SUBROUTINE UPLOAD(r15,r16,qfilnam,filtype,exec)
C
C     ACTUAL RECEIVING OF FILE.
C     DISCARD LAST FRAME.
C     CHECK FOR 1 RECORD FILE.
C     SEND BACK ACKNOWLEDGE MSG.(2nd type field=FF hex).
C     IF RECEIVE BAD FRAME, DO NOTHING BUT WAIT TO RECEIVE
C     SAME FRAME AGAIN.

      character*28    msg1/' Received successfully.    '/
      character*28    msg4/' Open file fail! Try again.'/
      character*28    msg5/' Type "sendfile FN.FT".    '/
      integer*2       iosb(2),done,filtype,exec,max
      integer*4       nichan, sys$qiow, sys$assign,recsize
      include         'edra0:[nossys.interlan]nidef.for'
      include         '($iodef)'
      byte            pad(100),getrec(512),getrec1(1024)
      byte            Rpacket(150),qfilnam(40)
      byte            Tpacket(130),r15, r16

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Print out:
      type *,' Request receiving of file'

C     Print filename:
      write(6,44)(qfilnam(k),k=1,15)
44    format(' ',15a1)

      type *,' from MDS system at address: 02 07 01 00',
     1       r15,r16

C     Initialize flag:
      done=0

C     Determine record size by file type:
      if (filtype.eq.1) then
            recsize = 128              ! text file
      else if (exec.eq.1) then
            recsize = 512              ! executable file
            max = 5
      else if (exec.eq.2) then
            recsize = 1024             ! EOD exec file
            max = 9
      end if

C     Open new file:
      if (filtype.eq.2) then
            open(name=qfilnam,unit=2,
     1           organization='sequential',
     2           type='new',carriagecontrol='list',
     3           iostat=ios,err=7,
     4           recordsize=recsize,recordtype='fixed')
      else
            open(name=qfilnam,unit=2,
     1           organization='sequential',
     2           type='new',carriagecontrol='list',
     3           iostat=ios,err=7)
      end if
```

```
          goto 90
7         call sendmsg(r15,r16,msg4)
          type *,' Open file fail! Try again'
          return

C     Receive file:
90        call sendmsg(r15,r16,msg5)
          type *,' Send instruction message.'
          type *,' Ready to receive file......'
          k = 1
20        istat=sys$qiow(,%val(nichan),
          1                %val(io$_readlblk),
          2                iosb,,,
          3                %ref(Rpacket),%val(150),,,,)
          if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
          if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
          if (Rpacket(18).eq.'0F'x) then   ! a 1 record file
                write(2,55)(Rpacket(j),j=19,146)
55              format(128a1)
                done=1
          else if (Rpacket(18).eq.'FF'x) then
                done=1
          end if

C     Send acknowledge message:

C     Assign destination address:
60        Tpacket(1)='02'x
          Tpacket(2)='07'x
          Tpacket(3)='01'x
          Tpacket(4)='00'x
          Tpacket(5)=r15
          Tpacket(6)=r16

C     Assign type field:
          Tpacket(7)='00'x       ! message
          Tpacket(8)='FF'x       ! acknowledge signal

C     Put data into transmit packet:
          j=9
          do i=1,28
                Tpacket(j)=ichar(msg1(i:i))
                j=j+1
          end do

C     Transmit packet:
          istat=sys$qiow(,%val(nichan),,
          1                %val(io$_writelblk),
          2                iosb,,,
          3                %ref(Tpacket),%val(136),,,,)
          if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
          if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
          type *,' Acknowledge is being transmitted......'

C     Load transmit data and send:
          istat=sys$qiow(,%val(nichan),
          1                %val(io$_ltds),
          2                iosb,,,,,,,,)
          if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
          if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
          if ((done.eq.1) .and. (k.ne.max) .and.
```

```
     1   (filtype.eq.2).and. (exec.eq.1)) then
                write(2,555)(getrec(j),j=1,(128*(k-1)))
                goto 70
          else if (done.eq.1) then
                goto 70
          end if

C     Write to record every after receiving 4 or 8 frames:
          if (filtype.eq.2) then
                do l=1,128
                     if (exec.eq.1) then
                       getrec(k*128-(128-l))=Rpacket(l+18)
                     else if (exec.eq.2) then
                       getrec1(k*128-(128-l))=Rpacket(l+18)
                     end if
                end do
                k = k+1
                if (k.eq.max) then
                     if (exec.eq.1) then
                       write(2,555)(getrec(j),j=1,recsize)
555                    format(512a1)
                     else if (exec.eq.2) then
                       write(2,777)(getrec1(j),j=1,recsize)
777                    format(1024a1)
                     end if
                     k = 1
                end if
          else if (filtype.eq.1) then
                write(2,666)(Rpacket(j),j=19,146)
666             format(128a1)
          end if
          goto 20

70        close(unit=2)
          type *, ' Receive completed!'
          return
          end
```

```
      SUBROUTINE SENDMSG(r15,r16,msg)
C
C     ACTUAL SENDING OF MESSAGE.
C     WAIT FOR ACKNOWLEDGE.
C     IF NOT ACKNOWLEDGE IN 5 SEC, RETRANSMIT.
C     IF NOT ACKNOWLEDGE IN 10 SEC, ABORT TRANSMISSION.

      character*(*)    msg
      integer*2        iosb(2),addr,done,count
      integer*4        nichan, sys$qiow, sys$assign
      integer*4        sys$bintim, sys$setimr, sys$waitfr
      integer*4        systime(2),time(2)
      include          'edra0:[npssys.interlan]nidef.for'
      include          '($iodef)'
      byte             pad(100)
      byte             tpacket(136),text(128),rpacket(150)
      byte             r15, r16

C     Assign a channel to NIA0:
      istat=sys$assign('NIA0',nichan,,)
      if(.not.istat) call lib$stop(%val(istat))

C     Assign destination address:
      tpacket(1)='02'x
      tpacket(2)='07'x
      tpacket(3)='01'x
      tpacket(4)='00'x
      tpacket(5)=r15
      tpacket(6)=r16

C     Assign type field:
      tpacket(7)='00'x        ! message
      tpacket(8)='00'x        ! don't care

C     Put data into transmit packet:
      j=9
      do i=1,28
              tpacket(j)=ichar(msg(i:i))
              j=j+1
      end do

C     Transmit packet:
      count=0
80    istat=sys$qiow(,%val(nichan),
     1               %val(io$_writelblk),
     2               iosb,,,
     3               %ref(tpacket),%val(136),,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
      type *,' Message is being transmitted.....'

C     Load transmit data and send:
      istat=sys$qiow(,%val(nichan),
     1               %val(io_eltds),
     2               iosb,,,,,,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))

C     Wait for 15 seconds and abort:
      call sys$bintim('0 ::15.0',time)
```

```
      call sys$setimr(%val(2),%ref(time),,)

C     Wait for 5 second and retransmit:
      call sys$bintim('0 ::5.0',systime)
      call sys$setimr(%val(1),%ref(systime),,)!retransmit
                                              !after 5 sec
C     Receive acknowledge:
      istat=sys$qiow(,%val(nichan),
     1               %val(io$_readlblk),
     2               iosb,,,
     3               %ref(Rpacket),%val(150),,,,)
      if(iosb(1).lt.0) call lib$stop(%val(iosb(1)))
      if(iosb(2).ne.0) call lib$stop(%val(iosb(2)))
C     Check the second type field if = FF hex:
      if (Rpacket(18).eq.'FF'x) then
              i = 19
              do while (Rpacket(i).ne.ichar('`'))
                      i = i+1
              end do
              write(o,66)(Rpacket(j),j=19,i-1)
66            format(' ',<i>a1)
              call sys$cantim( , )    ! cancel timers
              Rpacket(18)='00'x
              return
      end if
      if (count.ne.2) then
              call sys$waitfr(%val(1))
              count=count+1
              goto 80
      end if
      call sys$waitfr(%val(2))
      return
      end
```

APPENDIX D

VAX/VMS-MDS ETHERNET LOCAL COMMUNICATION NETWORK USER MANUAL

GENERAL:

ETHERNET is a program that will allow an individual to transfer a message or a file between VAX/VMS and the MDS System. After this program is executed on one of the VAX/VMS terminals, a user can leave the terminal and operate only on the MDS System terminal until he wants to disconnect the communication.

The user can execute this program only when he has LOG_IO privilege. If any problems occur while executing the program, user can contact one of the following VAX/VMS professional staff:

Albert Wong, Sp505, x2455
Olive Paek, Sp525b
Jeanne Bowers, Sp525a, x2168.

SPECIFIC:

ETHERNET.EXE program resides on the VAX/VMS under public user account with user name "INTERLAN" and password "VMS". Any user can copy this file by typing in the following commands:

```
$Copy <CR>
$From:      _dra1:[interlan]ethernet.exe  <CR>
$To:        *   <CR>
```

The user should also copy file ETHER1.EXE which should be executed after finishing the file transfer process in order to clear both transmit and receive buffers.

When these two files are copied, user can execute ETHERNET.EXE by typing

$r ethernet <CR>

then the message

Ready to receive message.....

will appear on the screen which tells the user that VAX/VMS has been connected to the Ethernet Local Area Network. After this point the user can do any file or message transfer by working on the MDS System terminal.

OPERATION ON MDS SYSTEM:

At the time this thesis is being written, there are 2 diskettes which contain program used to do file and message transfer between MDS System and VAX/VMS, one diskette is to be used with the MDS System which is connected to the single density disk drive and it contains the following programs:

LOGON1.COM
SENDMSG1.COM
SENDFIL1.COM

the other is to be used with the MDS System which is connected to the double density disk drive and it contains the following programs:

LOGON2.COM
SENDMSG2.COM
SENDFIL2.COM.

These two diskettes are now being used by Capt. Mark Stotzer USMC, the originator of the programs in the diskettes. Therefore, the final instructions on how to use the programs will be found in his thesis which will be completed by September 1983. However, the instructions on how to trans-

fers file or messages between the MDS System and VAX/VMS are as follows:

When the system is booted up with the above mentioned diskette, execute LOGON1.COM (or LOGON2.COM if you work with the double density disk drive) by typing

A>LOGON1 <CR>

To transmit a message from the MDS System to the VAX/VMS:

```
A>SENDMSG1 <CR>
ETHERNET CONSOLE MESSAGE TRANSMIT PROGRAM:
VERSION 1.11-SINGLE DENSITY: 06/10/83-MDS

NOTE:PROGRAM "LOGON1" MUST BE LOADED PRIOR TO
RUNNING THIS PROGRAM FOR PROPER OPERATION.
IF NOT-COLD BOOT AND TYPE "LOGON1" AND
THEN INVOKE THIS PROGRAM.

SELECT NET ADDRESS OF DESTINATION:
ADDRESS 00-04-0A( MDS SYSTEM ):ENTER 1
ADDRESS 00-03-EA( MDS SYSTEM ):ENTER 2
ADDRESS 00-07-7F( VAX 11/780 ):ENTER 3

3

INPUT MESSAGE(128 CHAR MAX)-END WITH ACCENT=> `

enter message here.`

SENT

A>
```

To transmit a file from your A disk to VAX/VMS:

Initiate SENDMSG like you want to transmit a message as above. But this time you must enter the special message as shown below:

A>SENDMSG1 <CR>

```
•
•
•
INPUT MESSAGE(128 CHAR MAX)-END WITH ACCENT=> '
@filename.filetype/txt'
SENT
********** RECEIVED MESSAGE IS:
Type "sendfile FN.FT"
********** END OF MESSAGE
CONNECTED TO THE ETHERNET-COLD BOOT TO DISCONNECT
A>SENDFIL1 filename.filetype <CR>
ETHERNET FILE TRANSFER PROGRAM:
VERSION 1.12-SINGLE DENSITY : 06/10/83-MDS
NOTE:FOR PROPER OPERATION, PROGRAM "LOGON1"
MUST BE EXECUTED FIRST TO LOAD THE INTERRUPT
HANDLER INTO MEMORY. IF NOT-COLD BOOT AND DO SO
SELECT NET ADDRESS OF DESTINATION:
ADDRESS 00-04-0A (MDS SYSTEM):ENTER 1
ADDRESS 00-03-EA (MDS SYSTEM):ENTER 2
ADDRESS 00-07-7F (VAX 11/789):ENTER 3
3
IS THIS A TEXT FILE [52.5KB MAX] ( Y OR N ) ==>
Y
READING THE FILE INTO THE BUFFER...
READ COMPLETE
********** FILE TRANSFER BEGINS **********
TX
```

```
TX
•
•
•
********** FILE TRANSFER COMPLETED **********

A>
```

To receive a file from VAX/VMS to your disk:

Initiate SENDMSG like you want to transmit a message as above. But this time you must enter the special message as follow:

```
A>SENDMSG1 <CR>
•
•
•
INPUT MESSAGE(128 CHAR MAX)-END WITH ACCENT=> `

!filename.filetype/exe`

SENT

********** FILE RECEIPT BEGINS **********
OPENING RECEIVE FILE: RECFROMX.NET
RX
RX
•
•
•
***** END FILE RECEIPT-SEE FILE RECFROMX.NET *****
CONNECTED TO THE ETHERNET-COLD BOOT TO DISCONNECT

A>
```

Note: Except for line spacing, the above sequences appear as they will at the MDS System terminal. The underlined items are those which you must enter (in proper sequence).

Important note:

1) Do not forget to end any message sent to the VAX/VMS with accent "`".

2) The "/txt" is used to indicate that the file to be transferred is a text file. The "/exe" is used to indicate that the file to be transferred is an executable file. User must specify this indicator correctly to yield the successful transferring of a file.

APPENDIX E

SYMBOLIC NAME FOR NI1010 CONTROLLER COMMAND CODE

This Appendix lists all NIDRIVER QIO function codes. The NI1010 Ethernet Controller User Manual contains a complete description of the NI1010 controller command codes and status returns.

```
C
C NIDEF.FOR - symbolic names for NI1010 controller command codes
C

        PARAMETER
        1 IO$$SMILM     = '42'X,        ! Set Module Interface Loop Back Mode
        1 IO$$SILM      = '82'X,        ! Set Internal Loop Back Mode
        1 IO$$CLM       = 'C2'X,        ! Clear Loopback Mode
        1 IO$$SPRM      = '102'X,       ! Set Promiscuous Receive Mode
        1 IO$$CPRM      = '142'X,       ! Clear Promiscous Receive Mode
        1 IO$$SROEM     = '182'X,       ! Set Receive-On-Error Mode
        1 IO$$CROEM     = '1C2'X,       ! Clear Receive-On-Error Mode
        1 IO$$OFFLINE   = '202'X,       ! Go Offline
        1 IO$$ONLINE    = '242'X,       ! Go Online
        1 IO$$ROBD      = '282'X,       ! Run On-board Diagnostics
        1 IO$$RRS       = '621'X,       ! Report and Reset Statistics
        1 IO$$RCDT      = '661'X,       ! Report Collistion Delay Times
        1 IO$$SRB       = '621'X,       ! Supply Receive Buffer
        1 IO$$LTD       = 'A20'X,       ! Load Transmit Data
        1 IO$$LTDS      = 'A60'X,       ! Load Transmit Data and Send
        1 IO$$LGA       = 'AA0'X,       ! Load Group Address(es)
        1 IO$$DGA       = 'AE0'X,       ! Delete Group Address(es)
        1 IO$$FRBQ      = 'C02'X,       ! Flush Receive BAR/BCR Queue
        1 IO$$RESET     = 'FC2'X        ! Reset
```

APPENDIX F

NIDRIVER FUNCTION AND STATUS CODE SUMMARY

This Appendix lists all NIDRIVER QIO function codes and IOSB status codes for each. The list of IOSB status does not include standard VAX/VMS error codes for device-independent errors.

The NI1010 Ethernet Controller User Manual contains a complete description of the NI1010 controller command codes and status returns.

Symbolic Function Code	Controller Function	Possible Contents of SecondIOSB Longword if status is SS$_NORMAL (In octal)
IO$_SETMODE! IO$M_STARTUP	Go Online	0
IO$_SETMODE! IO$M_SHUTDOWN	Go Offline	0
IO$_READLBLK	Supply Receive BUffer	0 (see note 1),17
IO$_READPBLK	Supply Receive Buffer	0 (see note 1),17
IO$_WRITELBLK	Load Transmit Data and Send	0,1,3,5,6,10,17
IO$_WRITEPBLK	Load Transmit Data and Send	0,1,3,5,6,10,17
IO__SMILM	Set Module Interface Loopback Mode	0
IO__SILM	Set Internal Loopback Mode	0
IO__CIM	Clear Loopback Mode	0
IO__SPRM	Set Promiscuous Receive Mode	0
IO__CPRM	Clear Promiscuous Receive Mode	0
IO__SROEM	Set receive-On-Error Mode	0
IO__CROEM	Clear Receive-On-Error Mode	0
IO__OFFLINE	Go Offline	0
IO__CNLINE	Go Online	0
IO__ROBD	Run On-board Diagnostics	See note 2
IO__RRS	Report and Reset Statistics	0,17
IO__RCDI	Report Collision Delay Time	0,17
IO__SRB	Supply Receive Buffer	0 (see note 1),17
IO__LTD	Load Transmit Data	0,5,17
IO__LTDS	Load Transmit Data and Send	0,1,3,5,6,10,17
IO__LGA	Load Group Address(es)	0,5,12,17
IO__DGA	Delete group Address(es)	0,5,12,17
IO__FRBQ	Flush receive BAR/BCR Queue	0
IO__RESET	Reset	See note 2

note 1: The STATUS byte of a received packet will contain none or more of the following octal error codes:

1 CRC error
2 alignment error
4 1 or more packets were previously missed

or it will contain:

17 Non-existent memory timeout on some buffer word (other than the first) occurred while DMAing the received packet into memory

note 2: The second IOSB longword at the completion of this command will contain one of the following diagnostic codes:

0 success
1 checksum error in local memory
2 NM10 DMA error
3 transmitter error
4 receiver error
5 loopback failure

LIST OF REFERENCES

1. Klinefelter, S.G., Implementation of Real-Time, Distributed Operating System for a Multiple Computer System, M.S. Thesis, Naval Postgraduate School, June, 1982.

2. Tanenbaum, A.S., Computer Networks, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

3. INTERLAN, Inc., Concise Ethernet Specification and NI1010 Unibus Ethernet Communications controller, 1982.

4. INTERLAN, Inc., NI1010A UNIBUS Ethernet Communications Controller User Manual (UM-NI1010A), 1982.

5. INTERLAN, Inc., INTERLAN NS2030 VAX/VMS (TM) Device Driver and Diagnostics User Manual, 1982.

6. Digital Equipment Corporation, VAX/VMS System Services Reference Manual, 1982.

7. Digital Equipment Corporation, VAX/VMS I/O User Manual, 1982.

8. Digital Equipment Corporation, VAX-11 FORTRAN Reference Manual and User's Guide, 1982.

9. Digital Equipment Corporation, DECnet-VAX User's Guide, 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Naval Postgraduate School Computer Technology Curricular Office Code 37 Monterey, California 93940	1
5.	Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
6.	Lieutenant Commander R.W. Modes, Code 52Mf Department of computer Science Naval Postgraduate School Monterey, California 93940	1
7.	First Lieutenant Thawip Netniyom, RTA Chulachomklao Royal Military Academy Rajadamnurn Ave. Bangkok, Thailand	2
8.	Captain Mark D. Stotzer, USMC SMC 2765 Naval Postgraduate School Monterey, California 93940	1
9.	Captain Ioannis A. Karadimitropoulos Delvinou 16, Papagou Athens, Hellas	1
10.	Captain T.F. Rogers, USN Box 327 Lumberport WV 26386	1
11.	Daniel Green (Code N20E) Naval Surface Warfare Center Dahlgren, Virginia 22449	1
12.	CDR J. Donegan, USN PMS 400B5 Naval Sea Systems Command Washington, DC 20362	1

13. RCA AEGIS Data Repository 1
RCA Ccrpcration
Government Systems Division
Mail Stcp 127-327
Moorestown, New Jersey 08057

14. Library (Code E33-05) 1
Naval Surface Warfare Center
Dahlgren, Virginia 22449

15. Dr. M.J. Gralia 1
Applied Physics Laboratory
Jchns Hopkins Road
Laukel, Maryland 20707

16. Dana Small 1
Ccde 8242
NCSC, San Diego, California 92152

202582

Thesis
N424
c.1

202582

Thesis
N424 Netniyom
c.1 Design and implementation of software protocol in VAX/VMS using ethernet local area network.

21 MAY 84 30132
MAR 16 85 33106
APR 30 [illegible] 33078
10 MAR [illegible] 33158
21 OCT 86 30473

202582

Thesis
N424 Netniyom
c.1 Design and implementation of software protocol in VAX/VMS using ethernet local area network.

thesN424
Design and implementation of software pr
3 2768 001 89915 6
DUDLEY KNOX LIBRARY